Design and Control of a Hybrid Power Supply

by

Francisca Muriel Daniel

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Electrical) in the Faculty of Engineering at Stellenbosch University

Supervisor: Dr. A. J. Rix

March 2020

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: .March 2020

Abstract

Design and Control of a Hybrid Power Supply

F.M. Daniel

Department of Electrical and Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MEng (Elec)

March 2020

A hybrid power supply (HPS) is the combination of two or more power sources as one single supply. An HPS is ideal for off-grid areas to provide sustainable and stable energy to improve the quality of life for the users. Due to the stochastic and intermittent nature of weather-dependent power sources, combining these sources increases the complexity of the design and control of an HPS. The different configurations in this thesis consider PV-modules, batteries, generators and a limited grid connection.

To solve the design problem, a genetic algorithm (GA) is implemented. The results are compared with commercially available HOMER software to highlight the differences between the two design methods. Three objectives are considered as part of the optimisation: technical, financial and environmental. The GA assesses different equipment configurations and sizes to not only look for a viable option but also a feasible configuration of different power sources. The algorithm clearly shows how the addition of more power sources increases the HPS's capacity factor and decreases the overall financial costs of the plant. A trade-off analysis between the different configurations is done. The GA can be seen as more robust than HOMER as it allows for user-specified constraints. HOMER can only assess one type of component (PV-module, battery, etc.) at a time, rather than looking at various options of the component.

The control system is implemented using a model-free Q-learning reinforcement learning (RL)-based controller which is compared to two baselines, random action and rule-based. The RL-based control system has no prior knowledge of how the system interacts and only learns through reinforcements such

as penalties and rewards. An Internet of Things-approach is added to increase the efficiency of the controller by using weather predictions to aid the RL-controller. The RL-based controller did not outperform the rule-based controller but did show improvement over the random action controller. The results indicates that the RL control system successfully minimised the loss of power supply and optimised the costs by using as much PV as possible. RL-controllers can be used as a feasible means of controlling an HPS. IoT-based application increased the utilisation of the PV and reduced the loss of power supply. The IoT-based implementation did not outperform the rule-based controller, but showed that IoT-methods can be exploited to increase the efficiency of controllers.

Uittreksel

Ontwerp en Beheer van 'n Hibriede Kragstelsel

("Design and Control of a Hybrid Power Supply")

F.M. Daniel

Departement Elektriese en Elektroniese Ingenieurswese,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.

Tesis: MIng (Elek)

Maart 2020

'n Hibriede kragbron (HKB) is die kombinasie van twee of meer kragbronne. Landelike en afgeleë areas is ideale voorbeelde waar HKBe elektrisiteit aan die verbruikers kan verskaf. Bronne wat afhanklik is van weersomstandighede se energie-uitset is onvoorspelbaar en afwisselend. Dit bemoeilik die ontwerp en beheer van 'n HKB. Die verskillende komponente van 'n HKB wat in hierdie tesis oorweeg word is, onder andere, PV-modules, batterye, generators en 'n beperkte kraglynverbinding.

Vir die komplekse kragbronintegrasie is 'n genetiese algoritme (GA) geïmplimenteer. Die GA se resultate is vergelyk met die kommersiëel-beskikbare sagtewareproduk HOMER. Die optimeringsproses het drie doelwitte: tegnies, finansieël en omgewingsimpak. Dit ondersoek nie net die mees lewensvatbare opsie nie, maar ook 'n haalbare gebruik van verskillende kragbronne. Die resultate van die GA het duidelik aangetoon dat addisionele kragbronne die HKB se kapasiteitsfaktor verbeter, terwyl die totale finansiële koste verminder. Verdere analise van die verskillende HKB's is ondersoek om meer duidelikheid te gee oor die verskillende aspekte vir beleggers betrokke by die keuse van hernubare projekte. Die GA is meer robuust en buigsaam vir 'n HKB-ontwerp omdat dit addisionele verbruikersbeperkinge in aanmerking neem. In teenstelling, ondersoek die program HOMER net een komponenttipe (bv. PV-modules, batterye, ens.) op 'n slag, pleks daarvan om verskillende komponentopsies te oorweeg.

Die beheerstelsel is gebaseer 'n modelvrye, Q-leer versterkingsleer ('reinforcement learning') (RL) algoritme. Hierdie beheerstelsel is met twee maatstaafbeheerstelsels, ewekansige ('random') en reël-gebaseerd, vergelyk. Die

RL-beheerstelsel het geen kennis van die stelselinteraksie nie en die proses van leer is deur ‘n metode van sogenoemde ‘beloon-en-straf’. Die doeltreffenheid van die beheerstelsel kan verder verbeter word deur middel van ’n ‘Internet-of-Things’-benadering (IoT) deur gebruik te maak van addisionele inligting soos weervoorspellings. Die resultate van die reël-gebaseerde beheerstelsel het aangetoon dat dit beter as die RL-beheerder presteer het om die kragverliese en kostes te verminder. Verdere ondersoek van die RL-beheerder teenoor die ewekansige beheerder het getoon dat die RL-beheerder die kragverliese beperk het, asook om die bedryfskostes te verminder deur die hernubare kragbron optimaal te benut. Dus kan die RL-beheerder as ’n lewensvatbare beheerstelsel vir ’n hibriede kragbron aangewend word. Die gebruik van IoT het die verbruik van PV teenoor die alleenlik RL-beheerder verbeter. Sodoende is die kragverliese van die IoT-implimentering ook verminder, maar nie tot op die vlak van dit wat verkry is deur die reël-gebaseerde beheerstelsel nie.

Acknowledgements

Foremost, I would like to thank my supervisor, Dr Arnold Rix, for his consistent support, guidance and knowledge. He allowed this thesis to be my own. Without his invaluable contribution, guidance and expertise, this research would not have been possible. I would like to also thank my fellow research group for their guidance.

I express my sincere gratitude to the Centre for Renewable and Sustainable Energy Studies (CRSES) for providing me with funding and the opportunity to finish my Master's degree in sustainable energy studies.

I give my thanks to my friends and family who have been with my journey and providing me with love and support during my process of researching and writing this thesis. In particular, I would like to thank my loving parents, Jurgens and Thelma-Anne, my sister, Jeanne, and my dear friend, Gerhard.

Lastly, but most importantly, I give all the glory to God, for abundantly blessing me with much more than I deserve.

Contents

List of Figures

List of Tables

Nomenclature

Variables

α	Learning rate	[ ]
β	Solar altitude angle	[°]
β	Weight	[ ]
γ	Discount factor	[ ]
δ	Solar declination angle	[°]
ϵ	Gaussian noise	[ ]
θ	Incidence angle between sun and collector face	[°]
θ_S	Solar zenith angle	[°]
ρ	Ground albedo	[ ]
ρ	Pearson correlation coefficient	[ ]
Σ	Surface tilt angle	[°]
ϕ_S	Solar azimuth angle	[°]
ϕ_C	Surface azimuth angle	[°]
a	Action	[ ]
A	Surface area	[m^2]
DHI	Diffuse horizontal irradiance	[W/m^2]
DNI	Direct normal irradiance	[W/m^2]
E	Equation of Time	[minutes]
E	Energy	[kWh]
GHI	Global horizontal irradiance	[W/m^2]
H	Hour angle	[°]
I_{BC}	Direct beam irradiance	[W/m^2]
I_C	Total irradiance	[W/m^2]
$\bar{I_C}$	Total insolation	[kWh/m^2]
I_{DC}	Diffuse beam irradiance	[W/m^2]
I_{RC}	Reflected irradiance	[W/m^2]
L	Latitude	[°]

LPS	Loss of Power Supply	[kWh]
$LPSP$	Loss of Power Supply Probability	[%]
r	Reward	[ ]
n_{day}	Number of day	[ ]
N	Number	[ ]
s	State	[ ]
T	Type	[ ]

Subscripts

π	Policy
avg	Average
Bat	Battery
Gen	Generator
i	i-th data point
Inv	Inverter
max	maximum value
min	minimum value
PV	Photovoltaic Module
t	Time t

Abbreviations

ANN	Artificial neural network
API	Application Programming Interface
CF	Capacity factor
CPU	Central processing unit
DHI	Diffuse horizontal Iirradiance
DNA	Deoxyribonucleic Acid
DNI	Direct normal irradiance
FL	Fuzzy logic
GA	Genetic algorithm
GHG	Greenhouse gas
GHI	Global horizontal irradiance
GPU	Graphics processing unit
HEV	Hybrid electrical vehicle
HOMER	Hybrid Optimization Model For Electric Renewables
HPS	Hybrid Power Supply
IoT	Internet of Things

LCE Levelised cost of energy
Li Lithium
LO Local optimizer
LPG Load profile generator
LPS Loss of power supply
LPSP Loss of power supply probability
LSTM Long short-term memory
MAE Mean absolute error
MDP Markov Decision Process
ML Machine learning
MPP Maximum power point
MPPT Maximum power point tracker
MSE Mean square error
N Number
NN Neural network
NPV Net present value
NREL National Renewable Energy Laboratory
O&M Operational and maintenance
PID Proportional-integral-derivative
PSO Particle swarm optimization
PV Photovoltaic
ReLu Rectified Linear Unit
REDIS The Renewable Energy Data and Information Service
RES Renewable energy source
RETScreen Renewable Energy Project Analysis Software (Canada)
RL Reinforcement learning
RNN Recurrent neural network
ROI Return on investment
SA Simulated annealing
SAURAN South African Universities of Radiometric Network
SARSA State-action-reward-state-action
SG Solar-Grid configuration
SGB Solar-Grid-Battery configuration
SGBG Solar-Grid-Battery-Generator configuration
SGG Solar-Grid-Generator configuration
SOC State of charge

T Type
TRNSYS Transient Systems Simulation Program
TS Tabu search
WTG Wind turbine generator
yr.no Norwegian weather predictions website
ZAR South African Rand

Chapter 1

Introduction

Society is heavily dependent on energy to perform daily tasks and the consumption of energy will continue to rise. According to the 2016 Energy Information Administration study from the United States Department of Energy, global energy consumption will continue to increase with 28% between the years 2015 to 2040, whilst 77% of the produced energy is generated by fossil fuel sources [1]. The United Nations Population Division has predicted that the earth's population will rise to approxomitely 9 billion people by the year 2050. An increasing population will increase the consumption of resources and electrical energy demands [2]. As the dependency and need for energy generation increases, the risks of running out of fossil fuel sources becomes greater and thus also the need for alternative sources.

Energy is generated from two main categories, namely renewable and non-renewable sources. Solar, hydro, wind and biomass are categorised as renewable energy sources (RES) and are free and effectively infinite. Free refers to the resource availability, but not the equipment required to harness the source. Renewable energy generation reduces the levels of air pollution by contributing electrical power without emissions. Non-renewable energy sources are defined as nuclear and fossil fuels. Fossil fuels has been formed from organic materials exposed to heat and pressure for over millions of years. These fossil fuels are categorised as crude oil, coal and natural gas. Renewable energy cannot be depleted; hence the term renewable. Non-renewables are in limited supply and will eventually become unavailable for power generation [3]. As energy demand increases, the use of fossil fuels will have to be replaced with a more sustainable source. This is not only for the sustainability of producing energy but also for the reduction of pollutant emissions. In recent years, the implementation of renewable energy has become cost-effective and an increased drive to incorporate more renewables has been evident worldwide.

Renewable sources, specifically solar and wind, are weather-dependent, which causes an intermittency problem. Each of the two energy source categories has

pros and cons. However, the combination of different energy sources, whether renewable or fossil fuel, has its advantages. The capacity factor, defined as the ratio between the generated electrical energy output and the maximum possible electrical energy output over a specified timeframe, increases and thus the efficiency and consistency of the power is increased [4]. Renewable resources exploits the use of energy resources which are locally available and is also considered to be more environmentally friendly.

A hybrid power system (HPS) is the integration and combination of two or more power sources. Figure 1.1 shows the general layout of an HPS. By combining two or more power sources, the drawbacks of the one source are replaced with the advantages of the other(s) [5]. The predictability of a non-renewable power source, such as a generator, can be used to replace the weakness of a photovoltaic (PV) system where it can only produce power during daylight hours. This is how the HPS increases the overall capacity factor. Combining a renewable resource with a fossil fuel resource reduces the usage of fossil fuel. Not only does this reduce pollution emissions, but it also saves money. Instead of running a 24-hour generator plant, the PV can offset some of the fuel consumption. Wind and solar energy work in a complementary form and the hybrid set-up can generate more power reliably than individual solar or wind farms.

HPSs are more energy-dense. However, the design has to be altered according to the location and resource availability. By increasing the number of power sources, the cost of the operation is lower compared to an individual plant

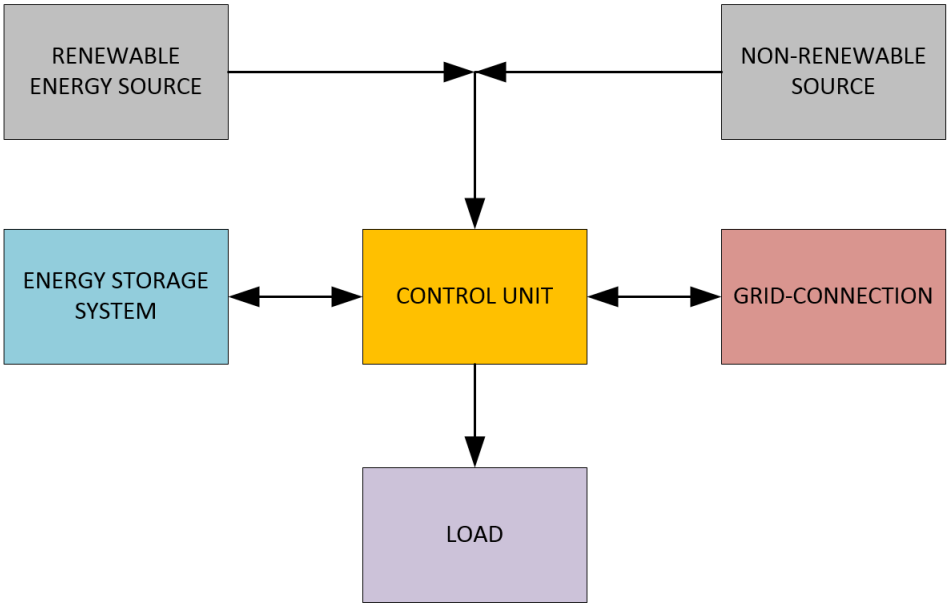

Figure 1.1: Hybrid Power Supply [6]

and overall reduces the installation cost. Thus the efficiency is increased and the cost of power over its lifetime is reduced [6]. HPSs with a RES has been deemed as the most appropriate for isolated communities, such as remote island or rural, off-grid or isolated areas [7]. The largest customer-base of HPSs consists of telecommunications companies, mine operators and remote rural communities.

HPSs can be utilised in a stand-alone approach, or be connected to the utility grid, which is known as a grid-connected approach. Stand-alone systems are usually in inaccessible areas where power transmission lines are not feasible to install. This can be because of the landscape, the right-of-way difficulties and/or environmental concerns. Even without these problems, transmission lines are still expensive to implement. RES-only stand-alone systems are subjected to production variations due to the weather. It would thus be advisable to incorporate some type of stored energy to supply the load when the source is unavailable, such as when solar energy cannot be generated during the night time or overcast days and wind energy when there is no wind [7]. Electrical energy is an important stimulant of the economy and daily life. New business opportunities, increased living standards, educational and health facilities with access to power can improve the overall quality of life in these rural or isolated areas. Grid-connected systems can bring an innovative aspect in the renewable power economy. Should an excess of generated energy occur, it can be fed back into the utility grid. In the event where the generated energy is not sufficient for the load, the grid can supply the shortfall. This can improve the overall feasibility and load availability of the renewable plant [7].

The integration of different power sources is currently still in its beginning phase and the global market size is relatively small. In 2014, the global hybrid grid-connected market was valued at $1.05 billion and is expected to reach $1.92 billion in 2019 [8]. For example, in Zambia, which is exposed to regular power cuts, mining companies have implemented a diesel-PV HPS to decrease its dependence on the utility grid [8]. In the South African context where load shedding is expected to become more prevalent, an HPS for mines could also be installed to reduce its dependence on the grid. This has already been done by the remotely-located Crominet chromium ore mine [8]. The mine added a PV plant to provide up to 60% of its power need, which aims to reduce fuel consumption. The nearly-completed Iamgold gold mine in Toronto also aims to reduce fuel consumption by adding a PV plant [8]. In Germany, close to the Swabian-Franconian Forest, a hybrid wind-hydro power plant is currently under construction. In Nevada, a renewable-only hybrid power plant consisting of geothermal, PV and solar thermal power generation was constructed [8]. The Danish city Aarhus integrated its entire renewable and heat generation units with existing conventional power systems to improve its energy self-sufficiency. In addition, it sells the surplus generated power back to the grid [8].

According to the South African Government website, two pilot HPSs have been initialised. The HPSs is situated in the Eastern Cape at the Hluleka Nature Reserve and the Lucingweni community [9]. Micro- and smart grids can also take advantage of the HPS to provide sustainable energy to its community.

Even with all the long-term environmental advantages of HPSs, two problems still remain: designing and optimising the equipment to produce competitively priced energy and creating a control system to interact between the different power sources and the load [8].

1.1 Design of a Hybrid Power Supply

An HPS integrates different power sources. However, there are constraints with the design of an HPS. Especially if RESs are incorporated, the location becomes important. Solar power would not, for example, be ideal for a very cloudy area and wind turbine generators (WTG) would be ill-suited for windless areas. Biomass, biogas and hydro plants would be better suited for areas with close access to these resources.

The practicality of the plant also has to be taken into account in terms of installation, operational and maintenance (O&M) and equipment safety costs. A historical profile of the load would be beneficial to improve the accuracy of the design process. If a load profile is not available, assumptions have to be made to aid the design process. When incorporating solar and wind sources, previous data is required to optimally design the HPS. Solar datasets, specifically for South Africa, are available from SAURAN (Southern Africa Universities of Radiometric Network), Solargis, GeoSun Africa and the South African government's energy website REDIS (The Renewable Energy Data and Information Service) [10–13]. If there is no data available for the load or the specified location, data has to be extrapolated from other sources with similar circumstances to aid the designing process.

Each energy system reacts differently in different environments; thus each system must be designed individually according to the investor's specifications. These specifications can be technical, environmental or financial. The performance indicators of an HPS include the investment capital cost, return on investment, consistency of supplied power, environmental impact and lifetime operational costs. An HPS must find a balance between the different performance indicators to find an optimal solution for the investor's specifications.

Designs formulated on the average or worst-case scenarios are inclined to produce oversized systems, which will increase capital expenditure and produce an unnecessary excess of energy [6]. The problem lies in the fact that the worst-case scenario tends to happen rarely. Also, the average values are not consistent [14]. If a solar source is used with great seasonal fluctuations, which

will produce a reasonable average value over 1 year, the design will result in a faulty design, which will waste money and resources. Other sizing methodologies have to be explored to produce better and more accurate designs of an HPS. These include software packages and computational algorithms.

The design method has to assess different configurations and power source integrations. Also, the data required to produce accurate designs are not always available. These data sources usually include the weather and load profile. A definition has to be given of what serves as an ideal HPS design for the specified area. It has to consider making an optimal profit for the investor while producing as much consistent energy possible and emitting minimal environmentally-harming pollution. All these factors increase the complexity of the design and control of a hybrid power supply.

1.2 Control of a Hybrid Power Supply

As the number of different power supplies is increased, the control system becomes more complex. The controller has to analyse the system in its totality, switch on and off different power supplies and shift energy to and from a storage system when incorporated. If the controller does not predict and navigate the load correctly, this can lead to a loss of power supply (LPS) and possible financial losses.

Different control methods include classic, hard and soft control [15]. The two categories under classic control are on-off and proportional-integral-derivative (PID) control. Gain scheduling-, state feedback-, optimal-, model predictive-, robust and non-linear and adaptive control all fall into the category of hard control methods. Soft control methods include fuzzy logic, artificial neural networks (ANN) and other evolutionary techniques.

Demand response methodologies consist of rule-based, model predictive and model-free controllers. However, a model-free approach simplifies the problem significantly, especially if the system is complex. A faulty system is produced if a model does not understand the process dynamics completely. Complex systems can be difficult to model accurately and thus extensive time has to be put in to understand the system dynamics. Model-free controllers do not need a model for the controller to function, however, model-free controllers require a large amount of data to learn and adapt to the system. As discussed in Section 1.1, if the necessary data is not available, the learning process of a model-free controller can be extensively prolonged and can take weeks, if not months or years for it to become a viable and sustainable option.

The definition of an adequate control system has to be critically defined for an HPS. As with the design methods, the control system must carefully balance the cost-effectiveness and power production, as well as storage, of the HPS. The

efficiency of the HPS controller can also be improved if it has prior knowledge of how the RES will act in the next few hours. A weather prediction unit can be incorporated to add this additional efficiency. However, these weather prediction models have to be trained and this can also become time-consuming, as well as require vast amounts of data. There are many weather websites available and thus information from accurate weather sources, such as yr.no, can be extracted in some manner. This information is extrapolated to predict the expected solar insolation for solar energy generation or wind speed for wind turbine generators.

1.3 Problem Statement

The combination of intermittent and stochastic resources, such as RESs, present a non-linear optimisation problem in the design of an HPS [4]. As the dimensionality of an HPS increases, the trade-offs between different performance indicators become an important factor in choosing the correct HPS design for the investor.

As more power sources become integrated, the control system becomes complex. The controller must assess the load and available power sources and make controller-decisions based on this analysis. If the system is not analysed correctly, switching between power sources can result in an LPS and/or poor cost optimisation.

1.4 Research Goals and Objectives

The research goals are defined to optimise the design of an HPS and to increase the efficiency of the control system. A computational algorithm can be modified for user specifications, which can thus aid the investor in their decision-making process. Should a control be able to predict what the load and power supplies will be with reasonable certainty, this information can be used to implement preventative measures in ensuring a consistent power supply with an optimised financial cost.

The research objectives are:

1. To optimise the design of an HPS by utilising a genetic algorithm (GA) as a tool to aid the investor's decision-making process,
2. To analyse the results obtained by the GA for a multi-attribute trade-off analysis;
3. To assess the validity of a computational algorithm by comparing it with commercially available software;

4. To create a control system which incorporates an RL-based algorithm to increase the system's effectiveness and efficiency;
5. Incorporate Internet Of Things (IoT)-based information to increase the control system's efficiency.

1.5 Thesis overview

The thesis consists of 5 chapters and 3 appendices.

In **Chapter 1** an introduction of the research is given by discussing the background of energy, renewable energy and the integration of different power sources. The advantages and disadvantages of HPSs are given, as well as examples of what has been implemented globally and locally. The design and control of an HPS is briefly discussed to give an introduction to the research goals and objectives.

Chapter 2 discusses the design and control of an HPS. Various design methods are discussed and the appropriate design, a GA, and validation method, HOMER software, is chosen. The fundamental background of the GA is explored, as well as previous work which has been done in this research area. The HOMER software package is briefly discussed to highlight why this is an appropriate validation method to compare to the GA. Several control strategies are discussed and a reinforcement learning (RL) using a tabular Q-learning method is chosen as a suitable control method for an HPS. The discussion of the previous research done with RL-based control for HPSs is given. A way to incorporate a solar prediction to the control system by using the Norwegian weather predictions website's (yr.no) Application Programming Interface (API) and a linear regression is examined, as well as a validation control method to compare the RL-controller. Two baselines are considered to compare to the control system: a random action and rule-based controllers.

Chapter 3 provides an overview of the experimental design. The design of an HPS using a GA is discussed, as well as the assumptions made when using HOMER. An RL-based control system is designed and is also improved by integrating an IoT implementation using available weather predictions from yr.no. The two baselines are used to compare the results of the RL-based controller.

In **Chapter 4**, the results of the research are presented. Firstly, the design of an HPS using a GA and HOMER is presented and analysed. Secondly, the control of an HPS using RL, as well as RL and IoT, is examined.

Conclusions are drawn in **Chapter 5**. A comparison of the GA and HOMER is discussed. A comparison between the RL-based controller and the baselines is discussed, as well as the RL and IoT-based controller compared to the original

RL-controller and baselines. A summary of recommendations and possible future work concludes the chapter.

In **Appendix A**, the clear sky model and calculations for irradiance and insolation are given. **Appendix B** discusses the data input used in the design and control of an HPS. The code for the implementations of the simulations is given in **Appendix C**.

Chapter 2

Reviewing system design and control methods

As briefly discussed in Chapter 1, the design and control of an HPS become complex due to the renewable energy location and the integration of different power sources. This chapter reviews the different design and control methods associated with an HPS. Suitable methods are selected, whereupon the relevant theory and fundamental background of these methods are discussed. Comparison tools and research contributions from other authors are reviewed to critically analyse what has been previously done. This chapter provides the relevant theory and background which may aid the reader's understanding of this thesis.

2.1 Design Methods

As mentioned in Section 1.1, there are constraints with the design of an HPS. Each energy system reacts differently in a different environment and thus each system must be designed individually. The integration of power sources creates a complex eco-system of energy exchange in the HPS. The design methods require data such as load profiles and weather information to increase design accuracy. The combination of inconsistent and variable resources causes the HPS design method to become a non-linear optimisation problem [4].

Before the design process can start, a measurement of how 'good' an HPS is, must be defined. These measurements are known as the objectives, which are essentially the goals of the design process. The design objectives can be environmental, technical and/or financial, depending on the investor. Different measures of HPS are expressed in Table 2.1. As more of these objectives are included, the design process becomes more complex. Trade-offs will have to be made between different objectives when a viable HPS is chosen. There are various methods of designing an HPS, including probabilistic, analytical,

Table 2.1: Measures of a Hybrid Power Supply Feasibility

Objective	Category	Description
Loss of power supply probability (LPSP)	Technical	The ratio between the load's loss of power and the total load over a time period, expressed as a percentage
Capital and lifetime costs	Financial	Investment cost and O&M costs incurred during its lifetime, expressed as a monetary value
Levelised cost of energy (LCE)	Financial	The constant price per unit of energy which will cause the investment to just break even, expressed as a monetary value
Battery state of charge (SOC)	Technical	Energy storage available in the HPS
Capacity factor (CF)	Technical	The annual operational hours of a system, expressed as a percentage.
Net present value (NPV)	Financial	The sum of discounted present values of incomes which is subtracted with the discounted present costs along the useful lifetime of the system, expressed as a monetary value
Solar power utilisation	Technical or Environmental	The total usable solar power used over the total usable solar power generated, expressed as percentage
Fuel usage	Environmental	The ratio between the fuel-based generated power and the total HPS energy supplied to the load, expressed as a percentage
Return on investment (ROI)	Financial	The investors annual return on their investment, expressed as a percentage

iterative and hybrid methods [14]. These methods will be discussed below.

Methods based on probability and statistics are the simplest sizing methodology. These methods are appropriate if long-term hourly data is unavailable. However, these methods are not an optimal solution. Renewable energy power sources vary seasonally and thus at least a year of analysis should be considered when designing an HPS. Probabilistic design methods are thus the easiest means of design because it requires considerably less data, but leads to inaccurate results. Inaccurate results can lead to an over- or underestimation in the design which in turn can result in unnecessary capital costs, in the case of overestimation, or an LPS when underestimated.

HPS design can be analytically done by representing the HPS as a computational model. This model will assess the HPS's feasibility by determining the performance of the system. This design method requires a large time-series database for accurate results. An evaluation of the HPS's feasibility can be done using software packages. Simulation tools available commercially are RETScreen, Hybrid2, TRNSYS and HOMER.

RETScreen is a Microsoft Excel-based spreadsheet model consisting of a set of workbooks. Each workbook models a specific power system configuration. The program determines the annual average energy flow and analyses the energy generation, life cycle costs and greenhouse gas (GHG) emissions. The software's objective is to reduce costs. RETScreen is associated with pinpointing and evaluating potential energy projects [16].

HYBRID2 is a simulation software developed by the National Renewable Energy Laboratory (NREL) and simulates an HPS with high precision calculations. The software does not optimise the system [17]. TRNSYS was developed by the University of Wisconsin and can simulate the system, but cannot optimise the design [17].

HOMER is a designing and analysing tool and is considered the industry standard for the design of HPSs [14, 18]. It incorporates several energy sources such as generators, wind turbines, solar PV-modules, hydro-power and battery storage, among others. The software is based on time-series models which predicts the hourly or minutely power system performance. HOMER determines in each step how the power equipment in the system is dispatched. The software determines how feasible the HPS configuration is, as well as also assessing the economic feasibility of the project [16]. The literature on the HPS design using the HOMER software has been produced by [19–21].

The design of an HPS is a multiple objective problem such that several objectives, being technical, environmental and financial objectives, have to be considered simultaneously. Metaheuristic methods can be used to solve these particular problems. Metaheuristic methods are usually stochastic and mimic a natural or biological principle which can be used in an optimisation or search problem. These optimisation methods include simulated annealing (SA) and tabu search (TS), as well as iterative methods such as GAs and particle swarm optimisation (PSO) [22]. Iterative methods uses a recursive process to find the best design configuration according to the specifications. A discussion of the use of different design methodologies will be presented below.

The GA is a stochastic global search and optimisation technique which is based on the theory of evolution by natural selection, which was formulated by Charles Darwin in 1859. It is based on the process over which organisms change over time as a result of inheriting physical or behaviour traits. If the trait increases the species' chance of survival, the trait is carried over

through the next generation. If the trait decreases the species' chance of survival, most likely the species will die out and thus not carry over the trait to the next generation. The algorithm is generally robust in finding a global optimal solution in a multi-modal and multi-optimisation process [14]. The GA produces a list of viable options by producing a genetically superior population. In this case, the population would consist of viable and feasbile HPS design configurations. Various studies have been done designing an HPS using a GA [4, 7, 20, 22–25].

The PSO algorithm was first presented in 1995 by J. Kennedy and R.C. Eberhart and the algorithm was initially used for the predatory behaviour of birds flocking [14]. Each agent is influenced by its own flying experience and its neighbours and will constantly modify its flight direction and velocity until it ultimately reaches the global best position through the entire search space [26, 27]. Artificial neural networks (ANN) are optimisation methods based on the nervous system structure. The networks are part of the field of machine learning. The ANN has to be trained because the neural network adapts based on the data it receives. The ANN will then produce results based on its training.

Bio-inspired methodologies require considerable computational processing and can be adjusted in real-time. It can function without any prior knowledge of the relationships between different variables and can deal with non-linearities. The iterative methods can be built and incorporated in various programming languages or software. The easiest means of implementation will be in Python, because of its open-source network, on-line support and the extensive number of libraries. Alternatively, Matlab Simulation tools can be used. Hybrid optimisation methods increase results- and convergence time and is often the most powerful optimisation tool to design an optimal HPS [14]. Hybrid iterative methods combine optimisation techniques with two or threefold optimisation objectives, such as technical, financial and/or environmental objectives. These methods can be done combining GA, PSO or ANNs [14].

GAs search for a list of viable options, whereas PSO for one global optimum. Because of the complexity as a multi-objective problem, a list would be more acceptable to analyse the trade-offs between different HPS sizes and configurations for the investor. Each objective influences how the results will perform and thus a list of different options will highlight how each objective is optimised.

It was thus decided to use a GA as the appropriate design method to assess different configurations, sizes and design goals. HOMER is considered the industry standard in HPS design, based on the literature research. It was thus chosen to compare the impact of using industry-standard software and a computational optimisation algorithm. The fundamental background of the

GA, as well as the previous work using GA with HPS design, will be discussed in the next section.

2.2 The Genetic Algorithm

In this section, the fundamental background of the GA is given and the previous work using GAs as a design tool for an HPS is discussed.

2.2.1 Fundamental Background

The GA is a biologically principled optimisation algorithm which mimics the theory of evolution formulated by Charles Darwin. Species develop through a process called natural selection where small, inherited variations on its genetic code increase the individual population member's ability to compete, survive and reproduce. Inferior population members will die out as they will be unable to carry their genes into the next generation. A fitter population are produced over generations through reproduction. Figure 2.1 shows the flow diagram of the GA.

The reproduction is done through mixing the Deoxyribonucleic Acid (DNA) with two parents or through mutation of the DNA string. The GA navigates through a large gene pool, called the population, to find the optimal combinations of genes. In this case, the combination of genes represents the ideal HPS configuration. Each gene constitutes a variable which represents the size/number or the type of a specific component. These variables are randomly generated and will be discussed at a later stage.

As mentioned, the algorithm randomly generates population structures or chromosomes. The number of population members and generations are specified by the user. Each of these chromosomes has an encoding solution and the encoding is to the likes of DNA strings [4]. The general chromosome structure for a represented HPS can have the following format:

$$[T_{PV}\ N_{PV}\ T_{Inv}\ N_{Inv}\ T_{Bat}\ N_{Bat}\ T_{Gen}\ N_{Gen}]$$

This chromosome structure is used to briefly explain how the GA uses the DNA as part of its optimisation. The T refers to a type of either PV-module, battery, generator or inverter. The N refers to the amount of a specific component in the system. The types are represented by integers. An HPS can thus, for example, have N_{PV} amount of type n PV-modules. Type n would then be cross-referenced to a PV-module database containing the attributes the specific module. These attributes will include the name, associated brand, price, warranty and additional power ratings. The GA will thus have a database of

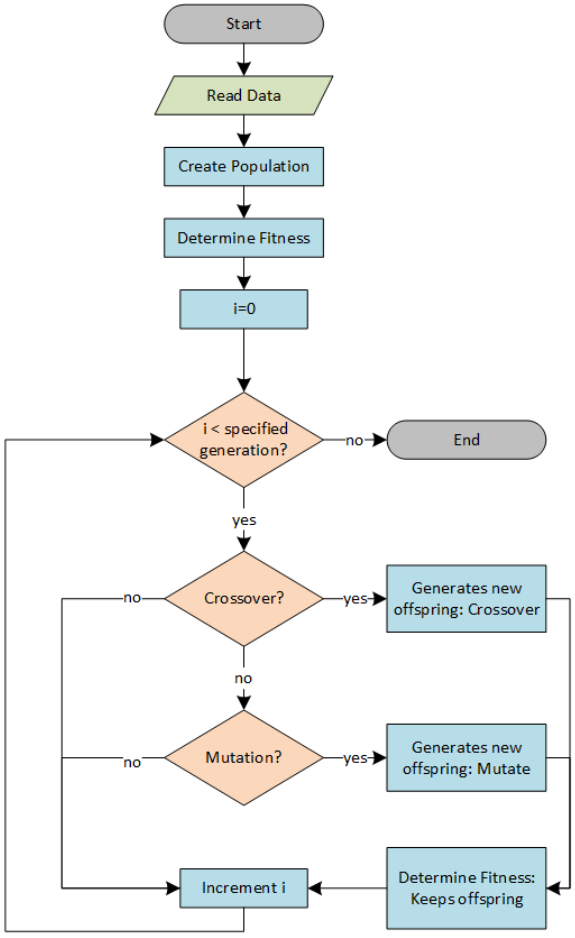

Figure 2.1: Genetic Algorithm Flow Diagram [6]

PV-modules, batteries, generators and inverters which is used for this cross-referencing.

Each DNA has information regarding the structure and combination of the HPS. As each generation progresses, the GA improves the population's overall fitness through the means of selection, crossover and mutation. Crossover and mutation is shown by Figures 2.2 and 2.3 respectively.

Selection duplicates the fitter structures and removes those with lower fitness ratings. Crossover recombines two parents' chromosomes to form a new chromosome [4]. Mutation creates new structures from one parent's structures by randomly altering the DNA of each structure. As shown as an example in Figure 2.2, each chromosome has a 50% chance of descending from one of the parents. The offspring is a combination of the two parents' genes. Figure 2.3 shows an example of how mutation occurs. One (or more) genes are randomly altered to create a new offspring. The offspring are evaluated on the fitness function. If the offspring proves to be fitter than the weakest population mem-

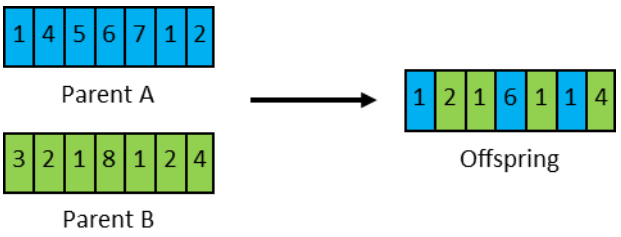

Figure 2.2: Crossover

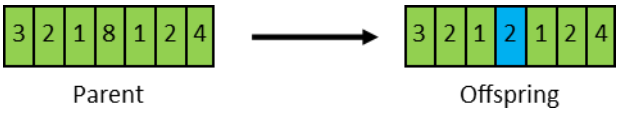

Figure 2.3: Mutation

ber, it replaces the weakest population member. However, if it is not fitter, it is discarded and the next generation starts.

Different selection strategies for crossover and mutation include tournament selection, proportional- and rank-based roulette wheel selection. These different selection strategies avoid premature convergence and increase diversity. A diverse population allows for different combinations and can result in better designs, which is ultimately the goal of the GA.

Tournament selection is a simple and efficient method of selection and is shown in Figure 2.4. It selects randomly chooses individuals from the population and these individuals compete with each other based on their fitness. The individual who has the highest fitness wins and is thus included in the next generation, whereas the weaker individual is not. The advantages of this selection method is that dominant population members will take over and the population will not require fitness scaling and sorting, which will reduce the computational time of the GA [28].

Proportional roulette wheel selection is the selection of individuals with a probability which is directly proportional to its fitness level. It can be visualised as a spinning roulette wheel and each segment of an individual is proportional to its fitness. The roulette wheel selection method is explained in Figure 2.5. The individual with the highest fitness level is likelier to be selected as a parent because it has a bigger segment of the roulette wheel. The advantage of this selection method is that it discards none of the individuals, as in tournament selection. Each of the population's individuals has a likelihood of being selected and the population's diversity is preserved. However, a bias can be introduced at the start of the search which may cause premature convergence and result in a loss of diversity [28].

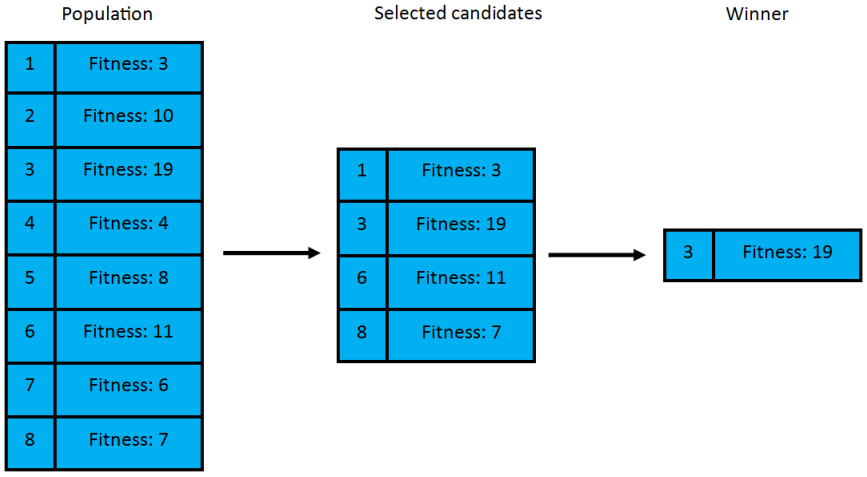

Figure 2.4: Tournament Selection

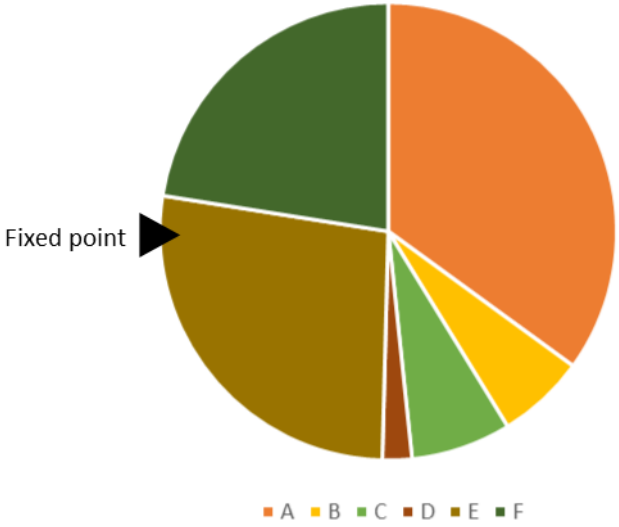

Figure 2.5: Roulette Wheel Selection

Rank-based roulette wheel selection is the selection methodology where the individual's selection probability is based on its fitness rank compared to the entire population. The selection method first sorts individuals according to their fitness in the population and then determines a selection probability according to its rank rather than fitness value. The method avoids premature convergence by introducing a uniform scaling in the entire population and eliminates the need to scale fitness values. However, it can be computationally expensive as a result of the continuous sorting of the population and also leads to slower convergence [28].

Additional strategies to increase population diversity is to reduce duplicates

or similar individuals. Early-stopping constraints can reduce computational time when a convergence arises.

Objectives are defined for the chromosomes, which will be the HPS in this case. The objectives for the HPS will have three aspects: financial, environmental and technical, as mentioned previously in Section 2.1. The algorithm's objectives will be discussed in more detail in Chapter 3.

As seen in Figure 2.1, the GA starts by reading the data input with regards to the weather and load profiles and a database of different components. These components are specified as the different power sources, such as PV-modules, generators, batteries, wind turbines, etc. These component sizes and types create the chromosome structure of the population member. Each population member is rated against a fitness function. This fitness function represents the population's members ability to meet the algorithm's different objectives. The fitness function is discussed in more detail in Section 3.1.4.4. The population member which can achieve as much of the objectives as possible will be seen as a fitter member. This member will have a greater chance of reproducing during crossover and mutation to create even stronger and fitter population members. Weak members do not achieve a good enough result from the objective's goal and its fitness rating will be lower. This means they will have a low probability of being selected for reproduction and a greater chance of being eliminated from the population of HPS configurations. The GA results in a list of various HPS combinations which are good on their own merits.

Initially, the GA randomly generates a population. Using crossover and mutation, more population members are created. Because the population number is fixed, weaker members are removed to make a place for the fitter members. The fitness is directly linked to how well the design objectives are met and thus a higher fitness rating is a more viable HPS solution. The resulting population is a fitter list of HPS solutions than when it started. These results can help the designer decide between different trade-offs, such as costs, power reliability, storage capacity and lifetime assessments. The decision of a viable HPS configuration is then left for the end-user to decide upon. The algorithm thus aims to highlight the different trade-offs between different configurations to assist the decision-making process.

2.2.2 Previous Work

There seem to be the two options for simulation purposes: a year-long hourly time-series set or lifetime approaches. A year-long hourly simulation was done in papers presented by [5, 23, 25, 29]. A lifetime approximation was done in papers presented by [7, 22]. For the 8760 hours, mathematical models are deemed adequate for the optimisation. The lifetime approximations are based on financial formulas and the data sets are usually average values or are assumed

to be constant. Both have advantages and disadvantages. The year-long simulation assumes that every year will be similar to the data set and thus the HPS will not be designed for yearly fluctuations such as droughts/unusual weather patterns. The lifetime approximations can be a problem if the real-life scenario tends to have extreme fluctuations. A mild temperature average could, for example, be the result of extremely hot summers and exceptionally cold winters. If the year's average temperature is used, this may not give an accurate insight into the design process. Over 20 years, the average values can be seen as sufficient for the algorithm, but the advantages and disadvantages of these two methods must be considered during the design. Detailed assumptions must be made to result in accurate designs.

In [20], the daily load is assumed to be constant and average monthly solar radiation and wind speed data is used during the assessment. The analysis was done for one year using these average values. Furthermore, a sensitivity analysis was done to optimise the system in different circumstances. The analysis was done with HOMER software, which produced the same results obtained by the GA. The similarity of the mathematical models is attributed to this fact. This paper notes that HOMER cannot simultaneously assess different component types. In this instance, a GA will provide faster and reliable solutions in the design and optimisation analysis of the HPS.

The objective functions vary greatly. Shahirinia et. al. aims in reducing diesel fuel consumption and minimising the total cost [7], whereas [29] seeks out to improve the grid stability by providing a buffer for the difference between the RES output and the load. Gonzalez et.al [5] aim to minimise the total life cycle costs and the metric of the fitness function is based on the NPV. In [23], the total costs are minimised, whereas Xu, et. al. [25] proposes to reduce the total costs which are constrained by the LPSP. In [4], a GA was constrained to stop when the objective function reached a pre-set target value of 240 kW. The objective function in the paper presented by Katsigiannis [22] aims to minimise the system's cost of energy. The objective functions presented in [24] are financial, technical and environmental. The financial objective minimises the life cycle cost. The technical objective maximises the energy supplied by the RES. The environmental objective minimises the annual GHG emissions. As the amount of objectives increase, the trade-off analysis becomes very important because one objective has to be constrained to a certain degree by another objective. The ideal HPS will have a balanced compromise between the different objectives for an overall better and viable HPS design. The objectives presented by Sopian, et. al. [20] maximise the use of the RES while minimising the use of the diesel generator. Thus these objectives can be seen as environmental and technical.

The design approach seem to differ for stand-alone and grid-connected HPS configurations. Grid-connected HPS [29] aids the grid for stability, whereas

stand-alone HPS are placed in rural areas to provide access to electricity. Stand-alone HPS [7, 23, 25] all include some kind of RES (wind or solar) and energy storage systems which is independent of a fuel source to produce power. The energy storage system varies from hydro-electrical pumps, to batteries and ultra-capacitors. A diesel generator or bio-gas/bio-mass plant is added to increase the HPS's ability to provide constant energy should the RES fail to provide as a result of the weather. A grid-connected HPS is proposed in by Kapfudza, et. al. where it highlights the difficulties which are faced in some parts of South Africa where communities live off-grid or are supplied by poor supply and services due to their geographical locations [4]. Furthermore, it implores the need to exploit other energy sources solve these problems.

The chromosome structures represent the different configurations of each HPS. In [25], the structure is [Type_{WTG} N_{WTG} Tilt N_{PV} N_{bat}]. The chromosome structure in [7] is [P_{dg} N_{PV} N_{wind}] and the battery capacity is predefined and not part of the optimisation. In the paper presented by Kapfudza, et.al, the chromosome's genetic encoding consists of the GHG emission index, economic index and the power output [4].

In [25] an additional optimisation is utilised to find the types and sizes of the components, after determining the PV-module and battery capacity. It then recalculates to the optimum fixed tilt angle of the PV-module. This optimisation can thus be seen as a hybrid algorithm because it incorporates a second and third optimisation. Similarly, Ma, et.al. proposes a two-step optimisation: firstly, the RES and storage system capacity size using a GA and secondly, a cost function to deduce the optimal combination of battery and ultra-capacitor size [7].

Atia, et.al. proposes a hybrid GA [23] which reduces the running time of the searching method. A secondary GA is implemented where five control set parameters are optimised. A local optimiser (LO) is also implemented in this paper. The LO was run only when the secondary algorithm reached the limited generation number or to interrupt the algorithm when no advance in the population was obtained for three successive generations. The hybrid GA produced a faster simulation time to make the searching time more viable.

In the paper presented by Shahirinia, et.al. several constraints were considered. These included the maximum running time of the diesel generator, the power delivered and stored by the battery bank and the hours in which the PV arrays generate power [7]. The selection method is roulette wheel selection where each population member's wheel slot is proportional to its fitness. Thus a fitter population member will have a larger chance of being selected. Ko, et.al, introduces a multi-objective optimisation design with various power sources in the HPS which optimises the size and configuration of hybrid cooling, heating, hot water and power systems consisting of RES systems and fossil fuel systems

[24].

The optimised design of the HPS is dependent on the design objectives of the researcher, investor and operators [24]. In the presented paper, the trade-offs are discussed to demonstrate how the investor will be able to decide upon an HPS [24]. This is another illustration of how the GA produces a list of viable HPS configurations and the most feasible option is left up to the designer and/or investor to decide. Gonzalez, et. al. [5] did a sensitivity analysis by setting a 10 % variation on the various financial and technical aspects, which will influence the results of the optimisation process. The sensitivity analysis shows how well the optimisation methodology reacts with changes with the input variables.

The grid-connected HPS can compensate for the loss of power supply (LPS) from the other sources and feed back the unused generated power to the utility grid. The stand-alone systems, which are ideal for off-grid and inaccessible communities should include some sort of energy storage and backup power source to compensate for the LPS from weather-intermittent energy sources. Currently, there is an incremental increase for cleaner and sustainable energy generation. The investors have to find a trade-off between investing in an HPS which can generate a return on their investment which will still consider environmental drivers. The technical aspects of the HPS can be seen as the most important because the HPS has to supply as much power as reliably and consistently as possible. The objectives should be clear and a measure of how the HPS performs should be assessed.

The one-year hourly simulation can provide accurate insights into the technical and short-term aspects of the design, but the lifetime projection gives more financial and environmental insights. It can be beneficial for the designer to look at both these options as an assessment of the feasibility of a project. An HPS with a very low LPSP may be financially unfeasible and a low capital cost HPS may have a very high LPSP and life cycle costs. Different components have greater O&M costs than others. Also, fossil fuel power sources may be subjected to future carbon tax laws or the depletion of the mineral source which can influence the design's results. As the number of power sources in the HPS increases, the complexity becomes considerably greater. In [22], six power sources are considered with a database of different types of power sources. The search time to find an optimal solution is considerably reduced by using a GA. The combination of every possible configuration and power source types results in 2.1 billion different combination outcomes. Simulating all these different configurations require approximately 234 years of simulations. This shows how significant the optimisation algorithm can perform in optimising the design of an HPS.

It is important to ensure that the correct simulation data set is chosen with

clear objectives of what the HPS must achieve. This thesis proposes an HPS design method of satisfying multiple objectives using an assessment of the hourly year analysis, as well as a projection of the lifetime to assess its long term feasibility. The results have to be compared to a baseline to assess how accurate the design method works and if there is an advantage or disadvantage in using the GA as a design method for an HPS. The chosen baseline is the HOMER software package and will be discussed in the next section.

2.3 HOMER Software Package

HOMER is a design and analysis tool for an HPS and contains a combination of standard generators, wind turbines, solar PVs, hydro-power and batteries. The software determines which dispatch strategy to incorporate given the power sources and determines its feasibility as an HPS. It also accesses the economic feasibility of the project by calculating capital, replacement, O&M, fuel and interest costs.

HOMER was used in [30] to design an HPS. The results were compared with a GA to show the correlation between the two optimisation techniques. It was noted that the GA had a lower cost of energy and NPV. Sopian et. al. also used HOMER to design an HPS [20]. The results were identical, attributing to the same mathematical models used in the GA as in HOMER. The paper notes that using HOMER can be a problem should different types of specific components be used to optimise the design. HOMER cannot simultaneously analyse different component types and thus every component type has to be assessed individually, along with its operational strategy. This can become very time consuming and thus for this type of purpose, a GA allows for a faster and more reliable optimisation method.

The most direct method to optimise an HPS is to use a complete enumeration method by using software such as HOMER. It ensures the best solution but can be proved to be extremely time-consuming [22]. The literature studies performed show conformity where in-house models appear to be conservative, flexible, better performance and simpler to use than the HOMER software [21]. From the literature, HOMER seems to be user-friendly and be able to provide accurate results. HOMER is also considered an industry standard and this can be a viable option to use as a design method.

2.4 Control Strategies and Methodologies

As the integration of different power sources increases, the complexity of energy exchange also increases. There is a fine balance between the power supplied to the load and to consider future load fluctuations, such as peak hours, a

limited grid supply, loss of renewable power generation or even load-shedding (in the South African context). The control system has to compensate for the loss of PV power when the overcast weather occurs or night hours, while still maintaining adequate back-up energy storage for rainy days or sudden spikes in the load profile. The energy exchange requires switching on and off certain power supplies based on the load and this switching time can cause delays in supplying power to the load. A control system has to either anticipate the LPS or be able to have an almost instantaneous release of power. The control system must be able to efficiently supply power by optimising battery life cycles and/or diesel generator usage. The control system must be able to analyse the generated solar power to decide if it should feed it to the load or charge the batteries, or both. Thus the control of an HPS is a highly complex energy exchange problem and different control algorithms will be discussed below to highlight the differences and advantages of said algorithms.

Before the analysis of different algorithms can begin, it is important to decide which measures will be used to determine how 'good' a control system is. These measures are very similar to the measures of the HPS design in Table 2.1. In addition, the efficiency, energy storage and reaction time of the control system can be considered. As mentioned in Chapter 1, different control methods include classic, hard and soft control [15]. Demand response methodologies include rule-based, model predictive and model-free control [31] and will also be discussed below.

The two categories under classic control are on-off and PID control. On-off control is simple to implement, however, the two states require the controller to make decisions based on multi-value variables. This results in a compromise between an environment which requires, for this instance, half of the power that the supply can provide. A high switching frequency of power supplies can lead to inefficient financial cost optimisation. PID control is a classic local-loop control and is easy to implement and test. The controller converts the error between the output signal and the input signal to action. It describes the error handling of the system. The proportional control attempts to achieve the output signal as fast as possible, whereas the integral control adjusts the summation errors to remove residual errors. The derivative control avoids adjustments which are implemented too fast for the system [15].

Hard control has six categories: gain scheduling-, state feedback-, optimal-, model predictive-, robust, and non-linear and adaptive control. If a system has several operational points, the gain scheduling controller designs linear controllers for each of the points. An interpolation strategy is applied to obtain an overall control of the system. Each of these controllers must be optimally tuned for the operational region and an increased controller number can result in time-consuming tuning jobs. The state feedback control assumes that the system environment can always be measured. A state feedback controller is

where the action of the control depends only on the values or external input. Optimal control is based on mathematical models. An optimisation problem is solved through the defined objective function. However, for real-time controls, complex systems can require considerable computational power and become inefficient for its purpose. Model predictive control has a predictive model, an objective function and a control law. A complex model, such as optimal control, requires expensive computational power. Robust control can handle time-varying model uncertainties and system associated non-linearities. Non-linear and adaptive controllers are developed through feedback linearisation and gain scheduling approaches. Adaptive control is a type of nonlinear control. The controller can modify its behaviour in response to dynamic changes in the process and can characterise the disturbances [15].

Soft control methods consist of ANNs, fuzzy logic (FL) control and other evolutionary techniques. ANN is a data-driven approach in a non-linear model where the optimal weights are determined which will minimise the errors between the estimated data and the target data. The network is trained to obtain the optimal weights and continues until convergence occurs. FL replicates human knowledge and reasoning in the form of membership functions and rules to control the inference actions for the control. FL does not require an exact mathematical model of the control process [15]. FL has proven to be more energy-efficient than rule-based strategies, but cannot be adapted automatically [32]. Similarly, dynamic programming (DP) can be used to optimise the rule-based control systems, but these cannot be applied in real-time [32].

The three most important demand response methodologies are rule-based, model-based and model-free (learning-based) [31]. Rule-based control strategies are based on a set of simple control methods consisting of if, then and while statements. The algorithms are relatively simple, less computationally expensive and in general, does not require forecasting. The algorithm is based on the domain and case-specific parameters which have to be tuned. This results in the difficulty of generalising these systems with a system that has different parameter setting [31]. Model predictive control uses mathematical models of the process dynamics. The strategies rely on accurate models which in turn produces more accurate results. The model dynamics have to modelled exactly. If the model for the process dynamics cannot be fully or partially understood, this can result in a faulty system model [31]. Model-free methods are data-driven, which requires large datasets. However, the system's dynamic model is not required because it is formulated when analysing and training the data. An example of a model-free control method is reinforcement learning (RL) where the learning curve comes from experiencing rewards or penalties based on the implemented action policy [32]. This results in less effort because no model is required and better generalisation for the application. In model predictive control, the strategies are case-specific and have to

be altered when the system environment experiences change [31].

RL is a computational approach where an agent learns from interacting with its environment. RL-based methods incorporate occupant behaviour and learn how the system interacts. RL does not require a model and a model-free approach can simplify a complex system considerably [15]. Data on consumption patterns, weather data and electricity generation becomes more easily available and thus increases the relevancy of data-driven techniques. The increasing computational power and data availability create an opportunity for these data-driven methods to be considered as an appropriate control method [33].

The RL agent receives rewards from its actions and learns if an action is beneficial for the environment to receive the maximum delayed reward. Said agent can balance the trade-off between exploring and exploiting different action policies [15]. RL can be a viable solution to control an HPS's energy exchange. The agent can learn to anticipate the load's profile and is rewarded if it executes a good policy or penalised when a bad policy is executed. Thus the controller's learning process is reinforced with rewards when correct or penalised if a policy results, for example, in a loss of power.

The controller's effectiveness can be increased if it can anticipate what the solar profile will be during the day. This means that some type of weather prediction can be incorporated. Various studies have been done to forecast the weather using machine learning techniques. For the scope of this thesis, it was decided not to build an algorithm which can predict the weather, but rather use available data from websites and use its information to extrapolate it to solar insolation prediction methods. The weather forecasting website was chosen as yr.no because of its user-friendly API. The solar power can be extracted using a simple linear regression algorithm.

An RL-based control algorithm was thus chosen to assess its validity as a viable control system for an HPS. An IoT-based solar irradiance prediction is also incorporated to assess how the controller's effectiveness can be increased if it also knows what to anticipate from the solar RES. Because the rule-based controller is the simplest controller to implement, it is chosen as a baseline comparison method for the control system. A random action controller will also be used as a comparison tool to the RL-based controller. The fundamental background and related work of RL-based control systems for hybrid power supplies are discussed below. An overview of the linear regression algorithm and the on-off controller is briefly discussed as well.

2.5 Reinforcement Learning

2.5.1 Fundamental Background

The two distinguishing attributes of RL are trial-and-error search and the delayed reward. The algorithm does not know which actions to take but rather discovers which actions or actions yield the greatest reward when performed [34]. The RL-model formally consists of a set of environmental states, a set of agent actions and a set of scalar reinforcement signals, which are typically 0 or 1 or a real number value [35]. The environment and action set are discrete. A learning agent interacts with the environment over time to obtain rewards [34].

The environment for this HPS will consist of the load, inverter, battery, generator and PV-modules. The agent learns how to interact between these sources and to optimally exchange energy. The rewards would be to, firstly, be optimal sustainable power supply which is the most cost effective. The cost optimisation refers to maximising the generated PV power, while minimising the use of generator fuel, as well as increasing the battery's longevity. In doing so, the feasibility of the HPS increases throughout its lifetime. This will be discussed in more detail in Chapter 3. The actions are penalised if the HPS does not have power to supply or poor cost optimisation. The value function will include long-term rewards for keeping constant power supplied.

The agent and environment interaction in a Markov Decision Process (MDP) is shown in Figure 2.6. The Markov property states that the future is not influenced by the past and only relies on the present and an MDP satisfies this property. The MDP models sequential decision making under uncertainty [36]. In Figure 2.6, state S_t and reward R_t are random variable outcomes after executing an action. The state $S_t \in \mathcal{S}$ refers to the environment's condition at discrete time step t.

The agent chooses a stochastic action $A_t \in \mathcal{A}$ in response to the environment. The agent then attempts to maximise future returns. The agent experiences an instant quantitative reward $R_{t+1} \in \mathcal{R}$ as the agent transfers to the new state S_{t+1}. The MDP is formulated as the sequence of the state, action and

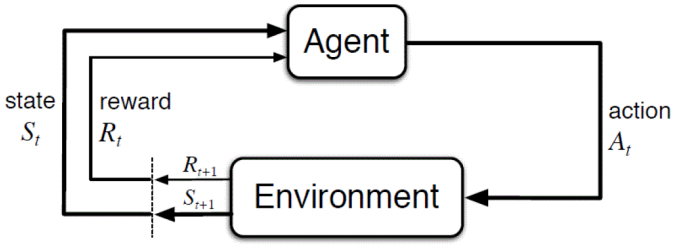

Figure 2.6: The interaction between the agent and the environment in an MDP [35]

reward. The states, actions and rewards will be discussed further below.

Equation (2.5.1) defines the joint probability density function for S_t and R_t. The distribution of the state and reward at time step t is dependent on the state and action from only the previous time step. Thus an action is executed on the previous state and transitions into a new state and this transition to the new state results in a reward. This dependency implies that the Markov property is satisfied and the transition matrix can be determined using (2.5.2).

$$p(s', r|s, a) = \mathbb{P}[S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a] \tag{2.5.1}$$

where $s, s' \in \mathcal{S}$, $r \in \mathcal{R}$ and $a \in \mathcal{A}$.

$$p(s'|s, a) = \mathbb{P}[S_t = s'|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} p(s', r|s, a) \tag{2.5.2}$$

RL differs from supervised learning. Supervised learning is where an external supervisor provides a training set of labelled examples. The system's objective is to generalise its responses to act correctly to states which are not in the training set. In the case where no examples are present, an agent has to learn from its own experience in different states [34]. The most important distinction between supervised and reinforcement learning is that no representation of input or output pairs is given for RL [35]. The agent is not given any prior knowledge of what will have the best long-term reward. Instead, all the information is gathered through experience about the possible system states, actions, transitions between states and the rewards obtained. On-line performance is also vital because the system's evaluation is often concurrent with learning [35].

There are four main subelements of an RL system, excluding the agent and the environment, namely the policy, reward signal, the value fuction and the model of the environment [34]. A learning agent must be able to observe its environment state and execute actions which would influence the state. It must have objectives which relate to the environment's state. The agent's job is thus to find a policy π which maps the states to actions and this policy will maximise some long-term measure of reinforcement [35]. The policy is stochastic and the probability of taking an action, a, in state s, is defined by

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]. \tag{2.5.3}$$

A policy, given by (2.5.3), defines the behaviour of the learning agent at a specified time. The policy maps the perceived environment states to the actions which are to be taken when it is in that specific state. The policy forms part of the core of an RL agent in which it alone is sufficient to determine

behaviour [34]. The policy is defining how an agent should act when it is in different states and is a distribution over actions given the states. The policy must be stochastic and can be considered as a function of actions. The policy can have the form of a look-up table or be presented as an approximation form.

The reward signal defines the goal of the RL problem. For every time step, the environment sends a quantitative reward to the RL agent, which signifies if the action had a positive or negative impact on the environment. The agent's only objective is to receive the maximum reward over the long term. Through trial-and-error, the agent starts defining which events are good and bad. This means that the policy may be changed if the action selected by the policy results in a low reward and changed to select another action which may return a higher reward [34]. The marginal distribution of the expected reward is given by

$$r(s,a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a). \tag{2.5.4}$$

A value function defines what is the best for the long run and can be defined as the total future accumulated rewards which an agent can anticipate to receive, should it start from its current state. Rewards determine immediate and inherent desirability of different environmental states. The value function takes into account the likely states which would follow and the rewards available in those future states. It can take into account an immediate low reward, but can still yield high rewards when followed by other states regularly and vice versa [34]. The state-value function defines $v_\pi(s)$, of a MDP, under policy π, as the expected of the return starting from state s is expressed by (2.5.5).

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right], \text{for all} s \in \mathcal{S} \tag{2.5.5}$$

The end goal of RL is to find a specified state's optimal policy. The optimal policy attempts to maximise the anticipated future rewards from time $t : G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}$, where $0 \leq \gamma \leq 1$ is the discount parameter. The state-value function $v_\pi(s)$ and action-value function $q_\pi(s, a)$ are two functional measures of RL which can be approximated from the data. The optimal policy π_* is formulated by evaluating either the optimal state function

$$v_*(s) = \max_\pi v_\pi(s) \tag{2.5.6}$$

or the optimal action-value function

$$q_*(s,a) = \max_\pi q_\pi(s,a). \tag{2.5.7}$$

Equations (2.5.7) and (2.5.6) makes use of the recursive relationships between the two sequentially ordered states or actions. Figure 2.7 (a) considers the

optimal state-value function when taking an action. Every possible executable action is evaluated by the agent. From this evaluation, the agent chooses the action with the maximum action-value. Figure 2.7 (b) assesses the dynamic and stochastic environment when an action is executed.

The model of the environment mimics the environment's behaviour which allows interferences to be made about how it will behave. Model-based methods have predefined methods and planning strategies, whereas model-free methods have no prior knowledge [34]. In the case where the environment or model is undefined, simulations over episodes can be done to estimate $q_\pi(s, a)$, under policy π with the expectation of the return starting at state s and taking action a, expressed by 2.5.8.

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right], \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A} \end{aligned} \tag{2.5.8}$$

In RL, a trade-off must be made between exploration and exploitation. Exploitation occurs when the learning agent exploits past experiences to obtain rewards. However, the agent also needs to explore the search space to improve its future selections. Both exploration and exploitation must be pursued and will result in failing at a task to learn from past mistakes. The learning agent must try different actions while still favouring those which appears to work the best. All RL agents have clearly stated goals, can observe the environment and execute actions which will influence the state of the environment. Also, it can usually be assumed that the learning agent has to operate from the beginning, despite the considerable uncertainty about the facing environment. The RL algorithm has to determine which capabilities are critical and non-critical when supervised learning is required. For RL to progress, it has to isolate and

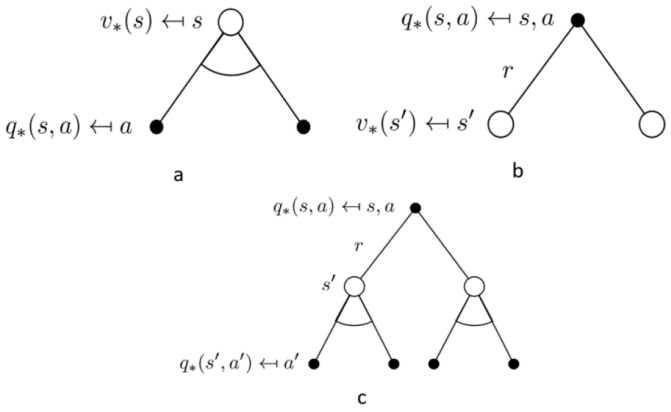

Figure 2.7: Backup diagrams for the optimal value functions [35]

study important subproblems. The agent uses experience and how well its actions performed to reach a goal to improve its performance over time [34].

The categories of RL algorithms are value iteration, policy iteration, value-based methods and policy search. The RL algorithm categorisation is summarised in Table 2.2. A brief overview of these different categories will be discussed below.

The estimations of systems with small and discrete state or state-action sets can be formulated using look-up tables with a single entry for each state or state-action value. The tabular Q-learning method is easy to implement and guarantees convergences. Q refers to the quality of the action, hence the term Q-learning. In Algorithm 1, the ϵ-greedy policy indicates that the agent chooses an action which has a maximum estimated action value with probability $1-\epsilon$. The update to a new action value is achieved by adding a so-called TD-error, $\alpha\left[R+\gamma\max_{a'}Q(S',a')-Q(S,A)\right]$, to the old action values. The value function $Q(S,A)$ (approximate value function) asymptotically converges to $q_*(s,a)$ (approximation target).

The parametrised value function approximation $q(s,a;\mathbf{w})\approx q_\pi(s,a)$ gives a mapping from the state-action to a function value. It generates state-actions which cannot be observed. Updating of the weight vector, $\mathbf{w}$, leads to the incremental method and the batch method. The incremental method updates

Table 2.2: Categorisations of reinforcement learning algorithms

	Model-based	Model-free
value iteration	Q-iteration	Q-learning
policy iteration	policy evaluation for Q-functions	SARSA
policy search	policy gradient	greedy updates

Algorithm 1 Q-learning Algorithm

Input: discount parameter γ; step size parameter α; $\{s,a\}\in\{\mathcal{S},\mathcal{A}\}$; $\epsilon>0$

1: **Loop** for each episode
2: Initialise $Q(s,a)$
3: **Loop** for each time step
4: Choose A by ϵ-greedy policy
5: Observe the immediate reward R and S'
6: $Q(S,A)\leftarrow Q(S,A)+\alpha\left[R+\gamma\max_{a'}Q(S',a')-Q(S,A)\right]$
7: $S\leftarrow S'$
8: **Until** S is terminal

Output: Q-table

the weight vector $\mathbf{w}$ by gradient descent given by

$$\mathbf{w_{t+1}} = \mathbf{w_t} + \beta \left[q_\pi(S_t, A_t) - \hat{q}(S_t, A_t; \mathbf{w_t}) \nabla \hat{q}(S_t, A_t; \mathbf{w_t}) \right]. \tag{2.5.9}$$

The learning rate $q_\pi(S_t, A_t)$ is iteratively obtained from the Bellman equation in (2.5.8).

The *SARSA(λ)* (State-action-reward-state-action) algorithm makes use of the approximation to update the eligibility trace and the value function. The linear case assumes $\nabla \hat{q}(s, a; \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s, a)$. The vector $\mathbf{z}$ has the same component number as $\mathbf{w}$ and is the eligibility trace. This holds a record of which components of the weight vector has already contributed to recent state values. The incremental method uses the experience only once to update the value function estimate and discards it in the next step. The batch method is sampling efficient and attempts to find the best fitting estimate of the value function to all of the data.

The fitted Q-iteration algorithm also uses gradient descent optimisation but updates $\mathbf{w}$ from a sample of observations. Tabular and approximation methods function in a value-based paradigm where the value functions have to be approximated. The policy is done using a greedy or ϵ-greedy strategy, while the policy-based method directly explores the parametrised policy using

$$\pi_\theta(a|s; \theta = \mathbb{P}[A_t = a | S_t = s; \theta_\mathbf{t} = \theta]. \tag{2.5.10}$$

The policy-based method presents better convergence, particularly for a continuous state-action space. The average reward for each time step is used as an objective function for episodic experiments. θ is iteratively updated using the gradient ascent technique. The deterministic policy can be avoided by assigning a probability to the action preference.

Because the HPS will be small and simple, the implementation will be done using the Q-learning algorithm. As mentioned before, the Q-learning algorithm is easy to implement, converges well and will be discussed below.

Q-learning is a value-based method to supply information to an agent of which action it should take [37]. The Q-learning agent does not require any prior knowledge of the environment and only requires which states exist and which actions are possible in each state [36]. The agent must have the ability to learn to take actions in a specific state which has a delayed reward. This means that although the agent might not receive an immediate reward, it will receive a larger reward later on [37]. All the states are thus infinitely visited and Q values are continuously updated until it converges [36]. In terms of the control system, the controller learns to optimally exchange energy in the HPS to ensure the load will be satisfied. As seen in Algorithm 1, the Q-learning

rule is defined. The neural network (NN) has to return a Q value for each possible action a in that specific state s. This Q value is defined as $Q(s, a)$ for all s and a. The $Q(s, a)$ is updated in training with the following rule in (2.5.11):

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(S', a') - Q(s,a) \right] \tag{2.5.11}$$

The new value of $Q(s, a)$ updates its current value by adding extra weights. The first weight is the r, which refers to the immediate reward which it will receive for taking an action a in state s. The second weight is the delayed reward calculation. The γ discounts the delayed reward's impact and is between 0 and 1. The $\max_{a'} Q(S', a')$ term is the maximum Q value which it can attain in the next state. Thus the agent starts in state s and takes action a and will finish in state s'. The code will then calculate the maximum Q value in state s' ($\max_{a'} Q(S', a')$) [37].

The $\max_{a'} Q(S', a')$ is vital to the RL algorithm because it is a representation of the maximum future reward the agent can obtain if it executes action a in state s. The discounted value γ prevents the agent of waiting forever for a future reward because the algorithm wants to collect the greatest award in the time. The $Q(s', a')$ implicitly holds the maximum discounted rewards for the state after it, i.e. $Q(s'', a'')$, and $Q(s'', a'')$ implicitly holds the discounted reward for state $Q(s''', a''')$, and so forth. This forces the agent not to look at the immediate reward, but also the possible future rewards which are discounted [37]. The α value is the learning rate during the updating and the current state is normalised by subtracting $Q(s, a)$.

The ϵ-greedy policy is used to aid exploration and exploitation. Initially, the agent must explore the problem space extensively to find a good local or global minima. Once the problem space has been adequately explored, the algorithm starts to focus on exploiting the knowledge it has obtained from the problem space. Thus the epsilon value allows for randomness in the action selection during the start of training. The ϵ is usually close to 1 at the start [37] and slowly decays to close to 0 during training. This slow decay allows a large exploration initially and then gradually exploiting the experience obtained. As it decays, the algorithm focuses on a good solution [37].

Because of the nature of RL, the risk of over-fitting the network is high. Usually, the gameplay is highly correlated and thus an action will most likely end with the same result. The addition of some kind of memory can be added to avoid over-fitting. All the data about the state, reward, action and the new state is stored in memory and this can be randomly sampled in batches to avoid over-fitting [37].

The controller will essentially be a randomised memory of previous steps and outcomes which the agent has experienced during training. The control system should hypothetically define policies which will increase the capacity factor of the HPS, while simultaneously optimising the cost-effectiveness and efficiency of the system. The controller can benefit from knowing what the foreseeable future solar power will be and use this as part of the control system.

A fundamental background on RL and Q-learning was provided to highlight the ways this can be used as a control system for an HPS. In the next section, the related works of HPS control and RL will be explored to study what has been done with these types of control systems. The implementation is discussed further on in Chapter 3.

2.5.2 Previous Work

The field of HPSs and RL control using Q-learning is still growing and thus very few quality literature works are available. However, previous work with regards to RL and hybrid electrical vehicles (HEV) [32, 38, 39], as well as building energy [31], has been done. Due to the lack of research material, these related works are also briefly discussed and interpreted to understand how RL control has been implemented.

Leo et.al. proposes a three-step-ahead Q-learning control algorithm for a micro-grid [36]. The micro-grid is grid-connected with a local consumer, a PV system and a battery storage facility. Mathematical models of the PV-modules and battery is used to calculate the power capabilities. The MDP considers discrete states and actions. The dynamic environment consists of the available solar power output and the consumer load. The algorithm looks ahead for three time steps and the rewards are quantitative performance measures. The agent's actions are influenced by the available solar power, battery level and load. The agent chooses the best 3-step-ahead planning based on the 3-step Q-learning algorithm. The long term objective of the consumer is to reduce the power consumption from the grid. The case study analysed the solar utilisation rate and the total grid usage as indicators of how the power consumption from the grid is reduced. The results showed that the Q-learning algorithm greatly decreased the grid power consumption and thus its objectives were achieved.

In the article presented by Mbuwir et.al. it notes that although Q-learning is a popular method, it throws away observation after every update, which leads to inefficient use of data [33]. Instead, a batch RL technique where the controller estimates a control policy based on a batch of its previous experiences is proposed. The microgrid consists of a PV system as a renewable source and a battery storage facility for a residential load. The state space has time-aware components which allow the agent to study the load profile.

The timing component considers the quarter-hour of the day and the day of the week. The state space also includes a controllable component with regards to the battery and an external feature, which is the observable external time- and weather-dependent information which cannot be influenced by the control actions. The three actions which can be executed by the RL agent are using no battery (i.e. use PV and grid power), charge battery using the PV while supplying the load with grid power and discharge the battery, as well as buying electricity if the PV and battery are not sufficient. The RL algorithm's objective is to maximise the generated PV power and in turn, minimise the use of the utility grid and the associated electrical costs.

A multi-agent RL algorithm is implemented by Bollinger and Evins [40]. The multi-agent RL algorithm allows agents to act independently to identify its optimal policy. The results indicated that the distributed multi-energy system was able to achieve near-optimal solutions if no storage was incorporated, but did not reach an optimal solution with storage. The results showed that the RL algorithm was unable to effectively deal with the trade-offs of storage scheduling [40]. Yue et.al. proposes RL-based dynamic power management for an HPS [41]. The system consists of Lithium-ion (Li-ion) batteries and a super-capacitor for mobile systems and using the RL algorithm, the efficiency was increased by 9%. The switching delay between power sources was also decreased using the algorithm. This is an indication that the RL agent can learn when to switch on or off certain power sources in anticipation of its usage. This can be advantages in systems which uses power supplies which takes longer to produce power, such as generators.

A deep RL solution for energy microgrids management is presented in [42]. Three discretised actions are described as the full rate discharge of the hydrogen storage, idling the hydrogen storage or charging it at the full rate. The ReLu activation function is used on all the layers except the output layer, where no activation is used.

The HPS in this thesis will contain a PV system, battery storage, generator and a grid-connection. Mathematical models of the PV-module, battery and generator will be used as part of the analysis. A batch Q-learning algorithm will be incorporated to maximise the use of memory and hopefully obtain a faster convergence to a minimum LPS with a maximum reward, while still optimising the cost of the system. It will be beneficial to rather use a single agent as part of the simulation process to ensure that the multiple agents do not clash. A simple rule-based-, as well as a random action-based controller, will be implemented as baseline validation of the RL-based controller. The controller will switch between the different power supplies based on the availability of power. A solar insolation prediction will be included in the RL-controller to analyse how the effectiveness can be increased if the agent has prior knowledge of the expected solar power generation.

2.6 Solar Insolation Prediction using Linear Regression

The Norwegian weather forecasting website, yr.no, has an API which is accessible using a python library. The weather prediction information from yr.no does not have information regarding solar insolation, but rather about the predicted clouds, temperature and wind speed. However, if a strong correlation between these prediction variables and the solar insolation can be found, it can serve as a vital information source for the RL-based controller. The relationship between these variables can be formulated using a linear regression which is integrated with the yr.no API and the controller. Regression is a technique for the estimation of relationships, of which the linear regression algorithm is the simplest to incorporate [43]. The linear regression algorithm can find a linear relationship between the different parameters received by the yr.no API and find a predicted value of solar power. This prediction can be used in the controller as additional information to increase the accuracy of the action policy execution. A similar approach to solar insolation prediction using a linear regression algorithm is presented in [44].

The general linear regression algorithm is given as

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_k x_k + \epsilon. \tag{2.6.1}$$

The solar insolation is thus modelled as a linear relationship between different variables which impact the solar intensity. β is a weight of the specific variable and the ϵ is the Gaussian noise. The mean square error (MSE) or mean absolute error (MAE) can be used to quantify the accuracy of the linear regression. The integration of the linear regression output and the RL controller may also become complicated if there is a scarcity of databases. To find a relationship, several prediction timestamps have to be used to review how far in the foreseeable future is appropriate to use. A database of these timestamps has to be built by scraping the yr.no website. If there is a disparity between the number of points in the scraped data and the RL-based controller, the integration of this information will become more complex. The seasonal changes affect the weather and thus at least a year's worth of data has to be scraped from the website to predict an accurate representation of the linear regression prediction.

The implementation is further discussed in Chapter 3.

2.7 Conclusion

This chapter gave an overview of different design methodologies, where it was decided to use a computational algorithm for the reason that it can be modified

to user specifications. A GA was preferred over a PSO because it produces a list of viable solutions, whereas PSO only produces one global outcome. HOMER software was discussed as a method of testing the validity of the GA. HOMER was selected because it is considered the industry standard in HPS design. A brief overview of the software package is discussed.

This chapter highlighted the different control strategies, whereupon it was decided to use an RL-based control system. A solar insolation prediction method using IoT is also incorporated. A rule-based and random-action-based controller will be used to test and validate the RL-based controller and a brief discussion is given of this controller.

Chapter 3

Designing systems and their control

This chapter discusses the experimental design of the two main components, being the design and control of an HPS.

In the two previous chapters, the relevant background of design methods was discussed, whereupon it was decided to use a GA as an appropriate design tool. A review of the fundamental background and the previous work associated with this design method was discussed. HOMER, a commercially available software, was chosen as a comparison tool to highlight the advantages and disadvantages of using a GA. The various control methods were discussed and an RL-based control was chosen as a suitable controller. Two baselines will be used to assess how well the RL-based controller performs. The RL-based controller is also given additional information with regards to the weather predictions to increase the use of the RES.

3.1 Design Using a Genetic Algorithm

The design of an HPS using a GA is based on Figure 2.1 in Section 2.2 and shows how the GA starts by reading in the data of the components, load profile and weather profile.

3.1.1 Data Input

The data used to design an HPS using a GA is discussed in this section. The data consists of a load profile, weather profile and a database of component types. The load and weather profile will be discussed below. The databases of the components are summarised in Appendix B under Section B.1 in Tables B.1, B.2, B.3 and B.4. These components were chosen based on the availability of the information.

3.1.1.1 Load profile

The load profile consists of 8760 hourly data points of energy loads. The weather profile consists of the average hourly direct normal irradiance (DNI), global horizontal irradiance (GHI) and diffuse horizontal irradiance (DHI) measured values for 8760 hours. The data is used as part of the analysis of the GA and will be discussed in more detail below. The location is based on Stellenbosch, because of the availability of data from the university and the load is based on a farm.

The load data consists of a date and the kWh usage for every half an hour. The data was acquired from a private consulting engineering company who worked on the farm and the use of the data for academic purposes was obtained. The peak load is specified as 80.67 kW with a load factor of 29%. The typical base load is 23.52 kW and the average daily energy used is 564.14 kWh. The data was processed to form 8760 hourly data points from the half-hour points, which is then used in the GA. The load was also analysed to aid in the design process. The period is from 1 January 2017 to 31 December 2017.

In Figure 3.1, the average hourly load per day of month is plotted. From the figure, it is clear that the beginning of the month had considerably greater average energy usage. The usage decays from 2nd until the 11th, whereupon it rises rapidly until it plateaus for the rest of the month. Figure 3.3 visualises the average monthly load which shows a significant spike at the beginning of the year and a lower use towards the end of the year.

Figure 3.2 shows the average 24-hour usage on the farm. The average hourly usage showed an increase in energy usage during the mid-morning hours until it plateaus to the late afternoon. The load decreases through the night. However, Figure 3.2 only considers annual averages and not seasonal fluctuations.

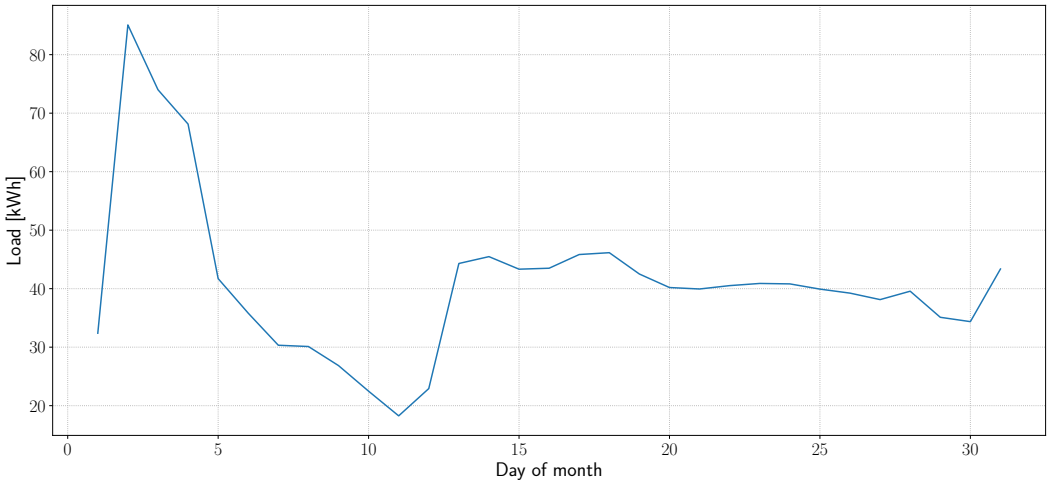

Figure 3.1: Design Average Hourly Load per Day of the Month

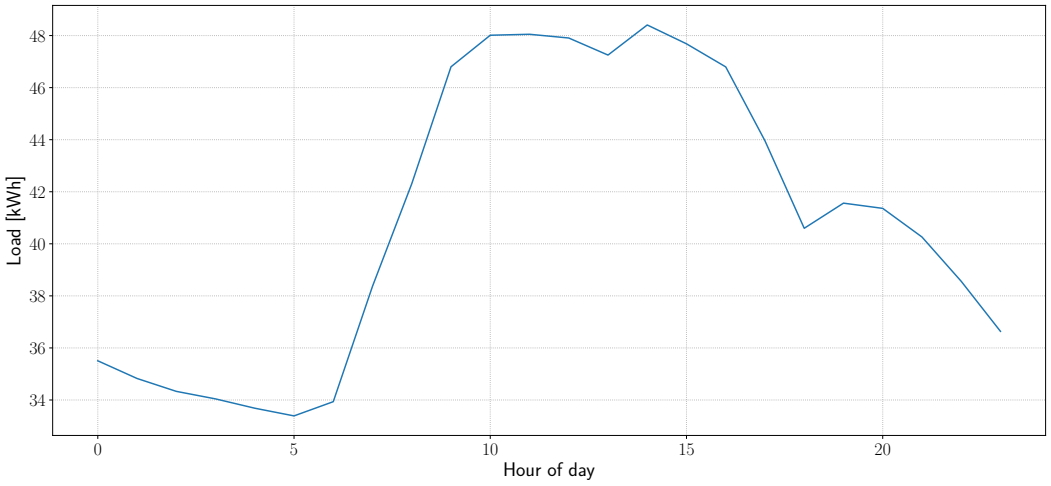

Figure 3.2: Design Average Hourly Load per Day

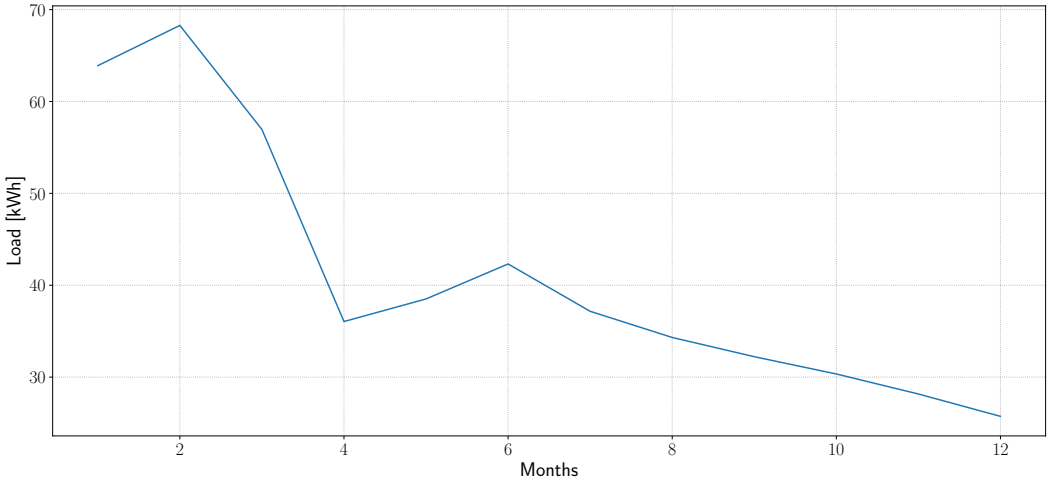

Figure 3.3: Average Hourly Load per Month

Because PV power will be considered as a power source, it is important to note how the load deviates over the seasons.

Figure 3.4 shows the entire data set over the course of 8760 hours. As expected, peaks can be seen in the beginning of most months and the load is considerably higher during the first 3000 hours of the dataset, reinforcing the deductions made from Figures 3.1 and 3.3. Figure 3.5 shows the average hourly load per day of the week, where 0 refers to Monday and 6 refers to Sunday. There seems to be a higher load on Monday and Sunday, and a slight plateau mid-week.

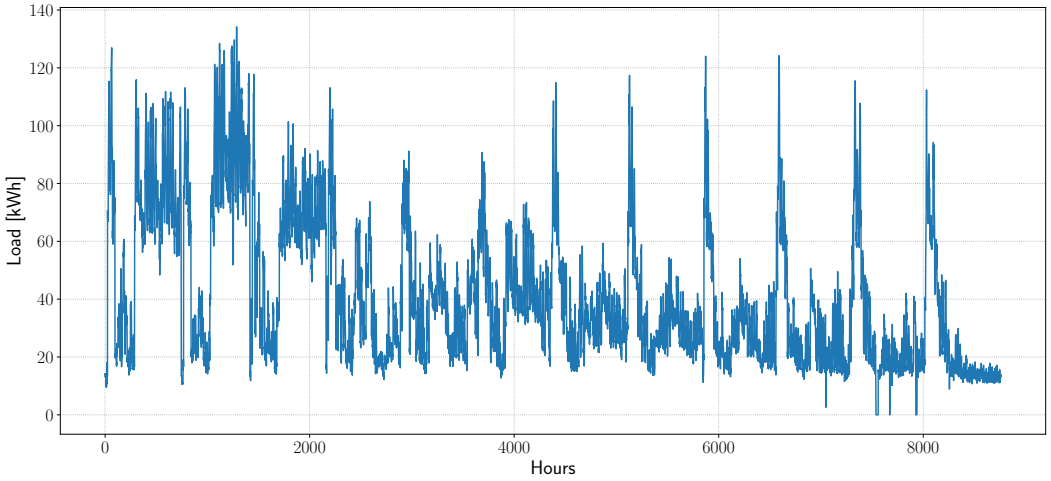

Figure 3.4: Design Hourly Load Over 1 Year

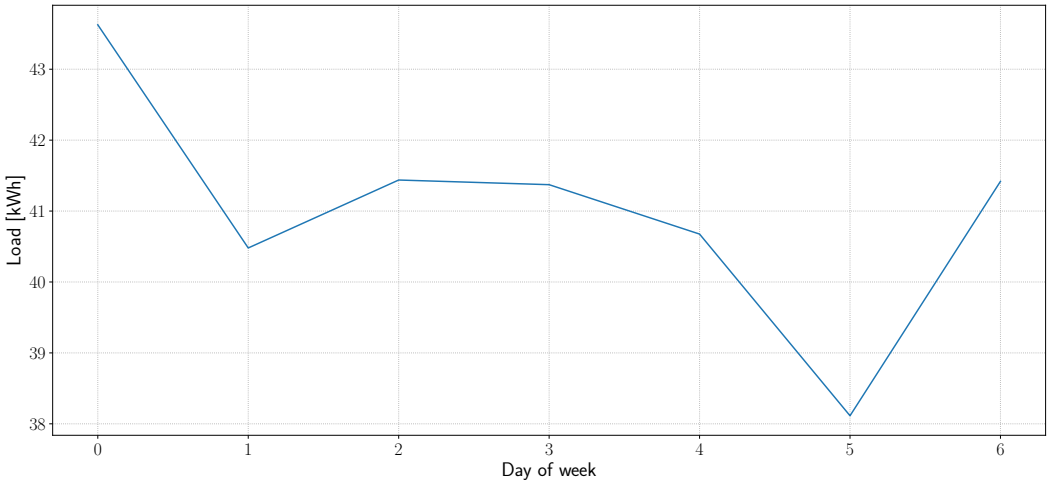

Figure 3.5: Design Average Day of Week Hourly Load

3.1.1.2 Weather Profile

The weather data was obtained from the SAURAN database [11] and for the same period as the load profile. The average monthly irradiance is shown in Figure 3.6. It shows a higher irradiance level during the summer months (October to February) and lowers during the winter months (April to September). The weather profile is ideal for the load profile during the early months of the year (January to March), however, the end-of-year summer months are not that suitable and will most likely generate an excess of solar power.

The initial analysis of the DNI, GHI and DHI is summarised in Table 3.1 and does not include night (i.e. non-daylight) hours data. Clear sky irradiance usually has a bell-curve where it peaks at the solar noon. The average solar

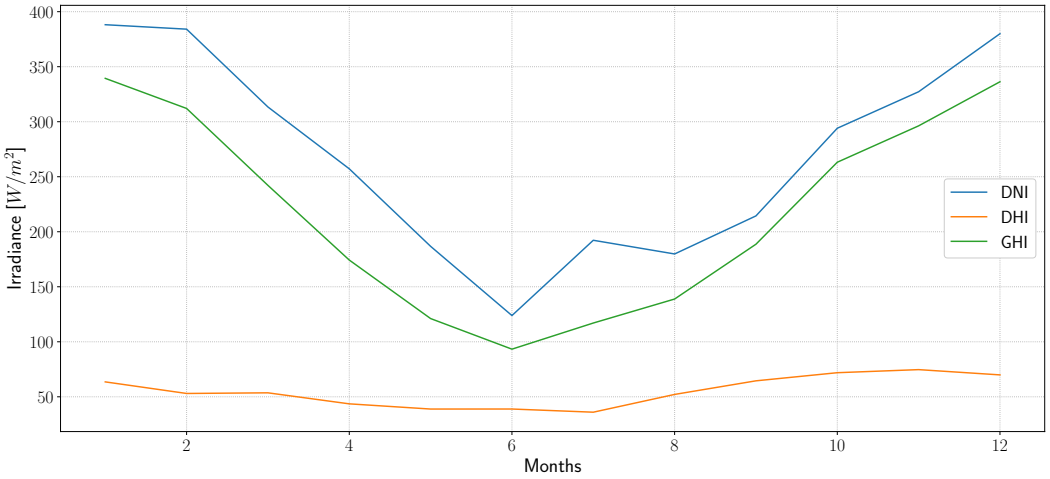

Figure 3.6: Design Monthly Average Irradiance

Table 3.1: Design Weather Statistics

Solar component	DNI (W/m^2)	GHI (W/m^2)	DHI (W/m^2)
Mean (μ)	448.8	359.4	91.9
Maximum	1072.4	1112.8	726.4
Standard deviation (ρ_X)	390.3	320.7	95.7

component shows a promising yield for a possible PV installation.

3.1.2 Power Source Mathematical Models

The power sources incorporated in the HPS configurations are PV-modules, battery storage, back-up diesel generator and a limited grid connection.

3.1.2.1 PV-modules

The energy generated by the PV modules [45] can be defined by (3.1.1):

$$E_{PV} = \bar{I_C} \cdot A \cdot \eta_{avg} \tag{3.1.1}$$

where $\bar{I_C}$ refers to the insolation on the collector in kWh/m^2, A to the surface area in m^2 and η_{avg} to the average efficiency of PV module. The average efficiency incorporates the system's derate factor, as well as the module's efficiency. The solar insolation is given by (A.2.12) in Appendix A. The calculations and further explanations for these values can be seen in Appendix A.

3.1.2.2 Battery

The charged battery energy storage in hour, t, $E_{B,t}$, is given by:

$$E_{B,t} = E_{B,t-1}(1-\sigma) + (E_{G,t} - \frac{E_{L,t}}{\eta_{Inv}}) \cdot \eta_{Bat} \tag{3.1.2}$$

where $E_{B,t}$ and $E_{B,t-1}$ is the energy stored in the batteries in hour t and the previous hour, σ to the self-discharge rate per hour of the batteries, $E_{G,t}$ to the available charge power, $E_{L,t}$ to the load demand in hour t, η_{Inv} to the inverter's efficiency and η_{Bat} to the battery's efficiency. The energy stored in the battery in hour t has lower and upper limits

$$E_{B,min} \leq E_{B,t} \leq E_{B,max} \tag{3.1.3}$$

where minimum allowable energy level $E_{B,min}$ is defined by Equation (3.1.4)

$$E_{B,min} = (1 - \text{DOD}) \cdot E_{B,max} \tag{3.1.4}$$

and $E_{B,max}$ is the maximum battery allowable energy level, which is assumed as the nominal capacity of the battery. The discharge and charging efficiency is assumed to be 90% and the depth of discharge (DOD) is chosen as 60% for optimal lifetime usage. Each battery is individually analysed throughout its lifetime usage.

3.1.2.3 Generator

The generator's fuel consumption is usually calculated using the fuel curve ratings. However, the precise fuel curves are not readily available and thus assumptions were made to account for this. To save computational power and memory, the generators are assumed on a 75% load when running. For the SGBG configuration, any unused generator power is fed to the batteries to charge. The SGG configuration disregards any unused generator power. A minimum run time of 2 hours is specified to avoid rapidly on-off switching of the generator. Each generator is assessed according to how much hours it runs and fuel consumptions.

3.1.2.4 Inverter

The inverter's model is based on Equation (3.1.5)

$$P_{out} = P_{PV} \cdot \eta_{Inv} \tag{3.1.5}$$

where P_{out} refers to the output inverted power, P_{PV} to the input power supplied by the PV-modules and η_{Inv} to the inverter's efficiency. The efficiency is included when inverting DC to AC power for the load.

3.1.2.5 Grid

Grid is a specified maximum hourly limit of 10 kWh with a reliability probability. The reliability probability accounts for unforeseen circumstances such as power outages and load shedding, should it occur in the near future. The reliability is specified as 99%. Thus for one year, 87.6 hours are assumed to represent off-grid scenarios. A random number is generated from a uniform distribution between 0 and 1. If the variable is less that 0.99, the utility grid does not fail. If it is greater than 0.99, it is assumed that the grid has failed for that hour and power cannot be supplied. The random variations adds a sensitivity and conservativeness to the design to account for possible scenarios the HPS can experience.

3.1.3 Assumptions

Certain assumptions are made for the algorithm. The lifetime assessment takes into account inflation and the O&M costs of the HPS over 20 years. The inflation rate is assumed as 5.0%. Table 3.2 shows the assumptions made in terms of the O&M and installation costs. These values are assumed to be a percentage of the capital cost of each component and thus linearly increasing or decreasing with regards to the system's size. For example, the O&M per PV-module is assumed to be 5% of the capital cost of the module. All O&M costs are assumed to be adjusted for inflation during the lifetime assessment of the system. The grid price is R1.45 per kWh

The fuel price is assumed as R16 per liter diesel and is adjusted with inflation each year. However, uncertainty holds with oil availability or escalating prices and is thus an important consideration for the feasibility of the HPS over its lifetime. The HPS does not sell any excess generated power back to the grid, which can also impact the ROI and feasibility of the HPS. Equipment was replaced on the year that warranty ends and the replacement costs is adjusted according to inflation. The replacement costs do not fall under O&M costs. The repairs and maintenance is assumed to be done during non-operational hours of the usage of the power source.

Table 3.2: Genetic Algorithm Financial Overheads and Operational and Maintenance Assumptions

Assumption	% of Capital Cost
PV maintenance cost	5
Battery maintenance cost	6
Generator maintenance cost	10
Installation costs	10

3.1.4 Genetic Algorithm Experimental Design

As briefly discussed in Section 2.2.1, the GA consists of population members with chromosome structures, selection, crossover and mutation, which occurs over a specified amount of generations, and the objective and fitness functions.

3.1.4.1 Chromosome Structure, Generations and Population

The chromosome structure is given as the types and numbers of four different components. These components are the PV-modules, battery, generator and inverter. The chromosome structure is presented by (3.1.6):

$$[T_{PV}\ N_{PV}\ T_{Bat}\ N_{Bat}\ T_{Gen}\ N_{Gen}\ T_{Inv}\ N_{Inv}\ ]\,. \tag{3.1.6}$$

The types can vary from 0 to the maximum number in the component database. The maximum number of components is predefined by the user to limit the search space. The type is randomly chosen according to a uniform distribution. Based on the different types, there are constraints to the different components. The PV capacity size is dependent on the inverter's ratings and the minimum and maximum value of the string number, as calculated by Equations (3.1.7) and (3.1.8). A random value between these minimum and maximum values are selected based on a uniform distribution.

$$\text{Min PV String number} = \frac{\text{Min. inverter MPPT Voltage}}{\text{PV MPP Voltage}} \tag{3.1.7}$$

$$\text{Max PV String number} = \frac{\text{Max. inverter MPPT Voltage}}{\text{PV MPP Voltage}} \tag{3.1.8}$$

The temperature effect of the PV-modules is neglected for the sizing purposes of the PV capacity. This is done to save computational power and the temperature effect assumed to average out throughout the simulation. When choosing a configuration, further investigation can be done to inspect the specific layout of the PV-modules and sizing to take into account the temperature effect.

3.1.4.2 Selection, Crossover and Mutation

In each generation, crossover and mutation take place. The probability of crossover occurrence is 80% and the mutation rate is 40%. Mutation creates an offspring by randomly altering chromosomes of a population member. Crossover creates an offspring from two parents in the population by randomly mixing the chromosomes. The algorithm proceeds by first examining whether crossover occurs with an 80% probability. Should crossover not occur, a mutation occurs with a 40% probability where each DNA has a 40% chance of

being altered. The selection of parents is based on the roulette wheel selection method, which is discussed in Section 2.2.1 of Chapter 2.

Theoretically, the GA can consider an infinite number of generations and population members to find an optimal solution. However, this is not ideal and will waste computational power. Thus another goal of the GA is to converge in the minimum time, while still maintaining a maximum diversity. Since the GA is a list of potential HPS configurations, the top fittest members will be considered as viable solutions. The population has to be big enough to ensure diversity, but not too large that it wastes computational power. The same can be said for the number of generations. As the population members become fitter, the criteria to surpass these fittest members become increasingly difficult. Thus after a number of generations, the algorithm's top members converge and rarely changes. Running the algorithm for many generations wastes computational power. An analysis of various populations with generations was done to determine the optimal sizes and the results can be seen in Table 3.3.

In Table 3.3, two measures of the population size and number of generations are given. The top member changes refer to when a new offspring replaces the best or fittest individual, whereas the offspring added column indicates that the offspring has replaced the weakest member of the population. The percentage is given as the ratio between the value and the number of generations. Even though the 120 population size with 160 generations has the same amount of top member changes as the 1000 population, 1000 generation combination, the percentage gives the perspective of how significant that number of change is.

The smaller populations have a higher percentage of top member changes compared to the larger population, whereas the larger population sizes indicate that the search space is saturated with an optimal solution early on in the simulation. In addition, higher generation numbers show lower top member change, which indicates that it is unnecessary to run a simulation for too long. The amount of offspring over the course of the simulation which is

Table 3.3: Population size and number of generation analysis

Population size	Number of generations	Top member changes	Offspring added
60	1000	58 (4.8%)	397 (39.7%)
120	160	6 (3.75%)	131 (81.8%)
500	500	4 (0.8%)	427 (85.4%)
1000	1000	6 (0.6%)	850 (85%)
60	80	8 (10%)	71 (88.75%)
60	160	13 (8.125%)	125 (78.1%)
2000	2000	2 (01%)	1759 (87.45%)
1000	2000	12 (0.6%)	1602 (80.1%)

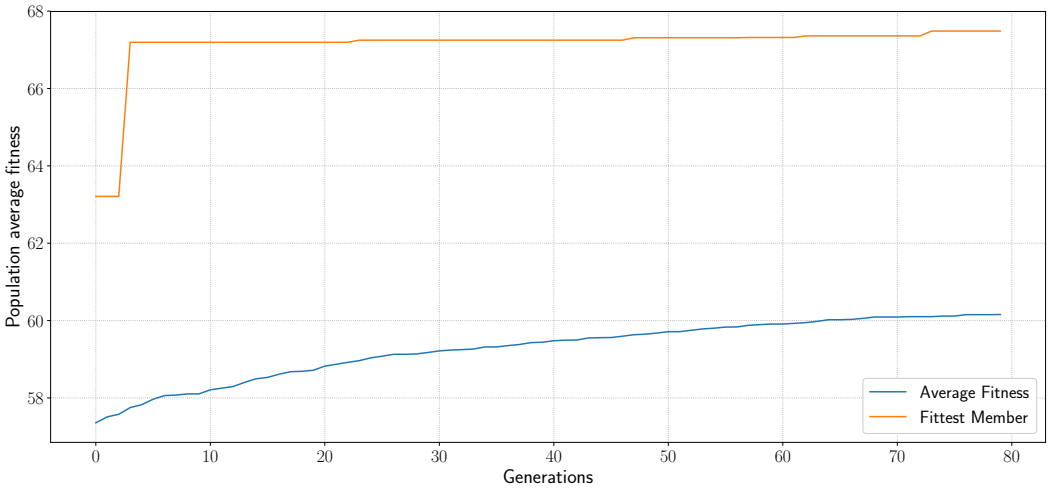

Figure 3.7: Average Population Fitness over 80 Generations

added to the group seems relatively high, except for the 60 population, 1000 generations combination. Figure 3.7 shows the average population fitness over 80 generations, as well as the top fittest individual of the population. The top fittest member increases rapidly in the initial generations, however, starts to plateau and becomes only marginally fitter.

Figure 3.8 shows the average population fitness over 1000 generations, with a population of 60, as well as the top fittest individual of the population.

In the figure, the average initially increases rapidly and converges as the generations continue. This is due to the weakest member being replaced with a stronger, fitter member, which will increase the population's overall fitness. However, as the overall population becomes fitter, the generated offspring has

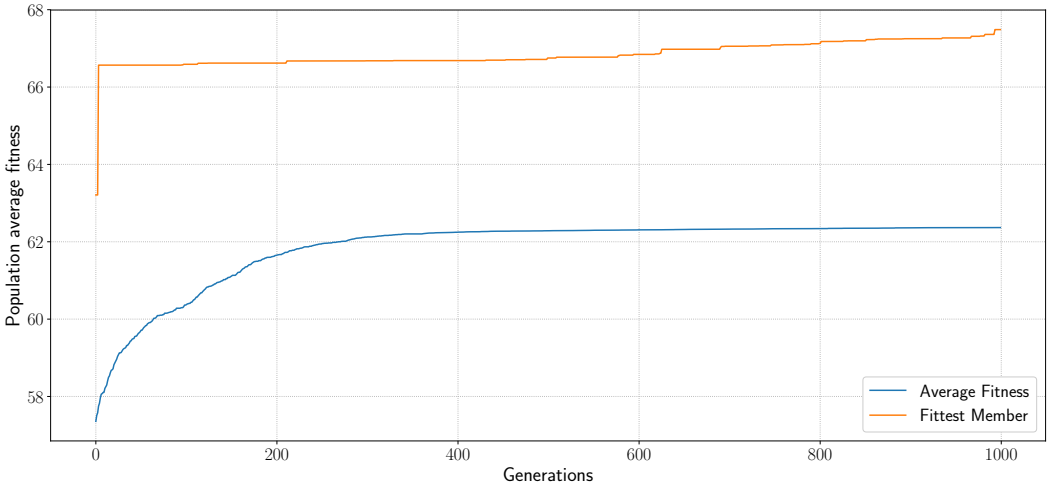

Figure 3.8: Average Population Fitness over 1000 generations

to become significantly fitter to be added to the population. This is indicated in the convergence on the figure where the average fitness of the population increases slower. The figure also shows the top fittest member and its fitness.

For this thesis, it was decided to use a population size of 60. The population is large enough to ensure enough diversity, but not too large that the algorithm's purpose becomes futile and defeats the overall purpose of the GA. The number of generations is chosen as 80, as most of the top fittest changes happen quite early on in the algorithm. A greater number of generations can marginally improve the HPS configurations, however, the differences become almost insignificant and will waste computational power. As the generations continue to crossover and mutate, the fittest individual becomes harder to beat and thus a large number of generations would not be beneficial to the algorithm.

3.1.4.3 Objective Functions

The different objectives highlights the trade-offs between different HPS configurations and power combinations. There are three main objectives, namely technical, financial and environmental and each of these objectives have different attributes. The technical objective aims to minimise the LPSP, whereas the financial objective aims to minimise the capital and life time costs by maximising the ROI for the investor. The environmental objectives aims to maximise the use of RES and to minimise the fuel usage, which attributes to pollution. The GA is defined as a multi-objective optimisation problem because it optimises more than one objective. By defining these objectives, an optimised HPS will be designed using the GA which produces consistent power, is financially feasible and will have a minimal impact on the environment. The GA evaluates four different HPS configurations: solar-grid (SG), solar-grid-battery (SGB), solar-grid-generator (SGG) and solar-grid-battery-generator (SGBG).

Technical objectives The LPSP gives an indication of how consistent the power produced by the HPS configuration is and is defined by

$$LPSP = \frac{\sum_{t=1}^{T} LPS_t}{\sum_{t=1}^{T} E_{L,t}} \tag{3.1.9}$$

where $LPSP$ refers to the loss of power supply probability, LPS_t to the loss of power supply in hour t and $E_{L,t}$ to energy load in hour t and

$$LPS_t = E_{L,t} - [E_{PV,t} + E_{G,t} + E_{B,t}] \cdot \eta_{inv} - E_{Grid,t} \tag{3.1.10}$$

where $E_{PV,t}$, $E_{G,t}$, $E_{B,t}$ and $E_{Grid,t}$ refer to the energy generated by PV-modules, diesel generator, energy stored in the batteries and grid energy in

hour t respectively and η_{inv} to the inverter's efficiency. The critical hour analysis of the LPSP assumes the LPS to be zero if the hour is specified as non-critical. The LPS is constrained never to be less than zero.

The capacity factor is given by the ratio of how many hours no loss of power supply occurred during the year and is given by (3.1.11).

$$CF = \frac{\text{Total hours running}}{8760} \tag{3.1.11}$$

Financial objectives The life-time costs and savings are based on the projection of capital costs. The O&M expenses are assumed to be a percentage of the capital cost, adjusted for inflation. The replacement is based on the end of the warranty and is also adjusted for inflation. Additional financial assumptions are also made with regards to inflation, fuel prices, fuel escalation and grid pricing. The investor's ROI is based on the amount of grid savings, the initial investment and the influence of the O&M costs.

The capital costs are defined by (3.1.12)

$$C_{HPS} = \sum C_{T_{PS,i}}\ N_{PS,i} \tag{3.1.12}$$

where i refers to PV-panel, generator, battery or inverter, C_{HPS} to the capital cost of HPS, $C_{T_{PS}}$ to the cost of component i and $N_{PS,i}$ to the number of component i.

The grid utilisation percentage is expressed by the ratio of how much of the load was supplied by the grid and is given by (3.1.13):

$$\text{Grid Utilisation} = \frac{E_{Grid,used}}{E_{Load}} \tag{3.1.13}$$

Environmental objectives To encourage the GA to produce environmentally-friendly HPS, an HPS which uses more PV power will be deemed as fitter, compared to a system which uses more generator power. The environmental objectives are dependent on the level of PV utilisation and the generator utilisation, which is expressed by (3.1.14) and (3.1.15).

The PV utilisation percentage is expressed by (3.1.14)

$$\text{PV Utilisation} = \frac{E_{PV,used}}{E_{PV,available}} \tag{3.1.14}$$

where $E_{PV,used}$ refers to the total PV energy used and $E_{PV,available}$ to the total PV energy available in 8760 hours.

The generator utilisation ratio is given as (3.1.15)

$$\text{Generator Utilisation} = \frac{E_{Gen,used}}{E_{Load,total}} \tag{3.1.15}$$

where $E_{Gen,used}$ refers to total generator used and $E_{Load,total}$ to total load in 8760 hours.

3.1.4.4 Fitness Function

The fitness function is given by (3.1.16). The fitness function is a weighted linear combination. Each weight of the different attribute indicates how important the investor views the objective and the fitness value indicates how well the overall objectives are met.

$$Fitness = \sum w_i Fitness_i \tag{3.1.16}$$

where i refers to the objective (technical, economical or environmental), w_i to the weight of objective i and $Fitness_i$ to the fitness of objective i .

Each attributes weight, w_i, is directly correlated to how important the investor views the objective. The different attributes for this GA are ROI, LPSP, critical-hour LPSP, and PV-, grid- and generator utilisation. The assigned weights can be observed in Table 3.4. The GA evaluates the new offspring created during mutation or crossover using the fitness function. Based on its fitness, the offspring will then replace the weakest population member. Should the weakest population member be fitter than the offspring, the offspring is discarded. This illustrates how the overall population becomes fitter with each generation. The technical objectives are assumed to have the highest priority and thus are attributed with the heaviest fitness weights. The financial- and environmental objectives are prioritised second and third respectively.

Table 3.4: Fitness function weights

Weight	Value
LPSP	12
Renewable energy utilisation	4
Grid utilisation	1
Generator utilisation	1
Capacity factor	3
Return on investment	1

3.2 Design Using HOMER software

HOMER is used to design an HPS. Assumptions with regards to the simulation can be seen in Table 3.5. The inflation rate was specified as 5% and the grid price is R1.45 per kWh, as with the GA. The project lifetime is 20 years and the generator has a minimum running time constraint of 2 hours.

The results obtained from this simulation is discussed in Section 4.1.2 of Chapter 4.

3.3 Reinforcement Learning-based Control System

3.3.1 Data Input

The load and weather profile consists of 525600 minutely data points during 2018. The location is, as with the GA, in Stellenbosch due to the accessibility of information. The control system has to be modelled as accurately as possible, without the waste of computational power. It was decided to use an interval frequency which has readily available data. A higher frequency, such as minutely intervals, can be manipulated to longer intervals, such as 5, 10 and 15 minute intervals. However, it is more difficult to extrapolate minutely intervals from 15-minute intervals. To ease the data manipulation process, the smallest possible available interval is considered and can thus be adjusted and scaled for further data processing.

The SAURAN website [11] has a scope of minutely-intervals and based on this, the controller's time interval was chosen as minutely. A load profile for this interval was not readily available and thus had to be artificially generated as accurate as possible. The Load Profile Generator (LPG) [46] is a modelling tool for residential energy consumption. It performs full behavioural simulation of a household and generates subsequent load curves.

Table 3.5: HOMER Assumptions

System component	Capital ZAR/kW	Replacement ZAR/kW	O&M ZAR/kW/Year	Lifetime years	Efficiency %
Battery	2,500	2,500	50	10	81
Converter	3,670.20	3,670.20	36.70	5	95
PV Module	7,115	7,115	355	20	20.4
Generator	3,249	3,249	0.20	20	40

3.3.1.1 Load profile

The household was specified as a family of five: Arthur, a 45-year-old male, and Cassie, a 40-year-old female, are the parents of three boys: Gareth (age 8), George (age 12) and Gregor (age 4). Each of these household individuals has a unique energy behavioural pattern which the LPG takes into account in the simulation. One parent works at home and the other at the office. It was also assumed that the family goes on holiday during the June recess. From this, a minutely load profile was generated and the profiles are summarised in Figures 3.9, 3.10, 3.11 and 3.12.

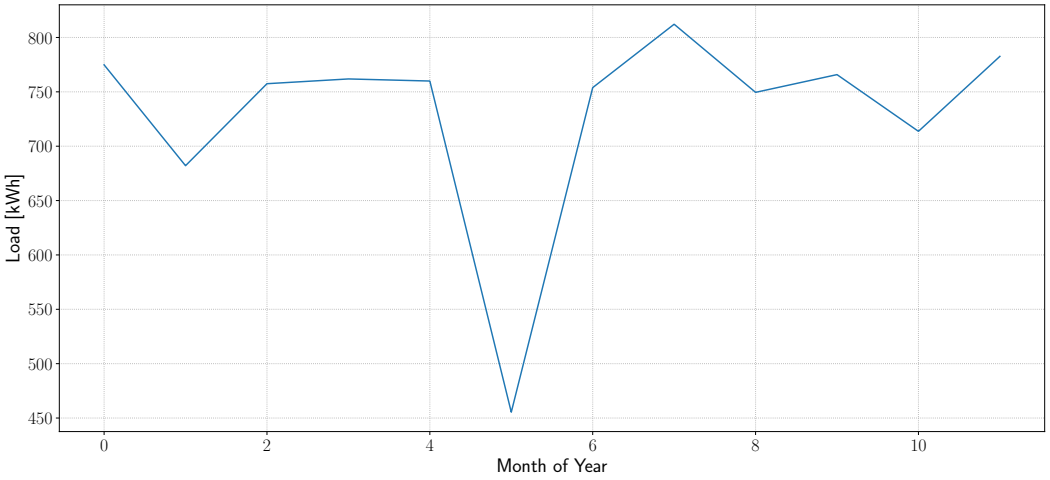

Figure 3.9: Controller Total Monthly Load

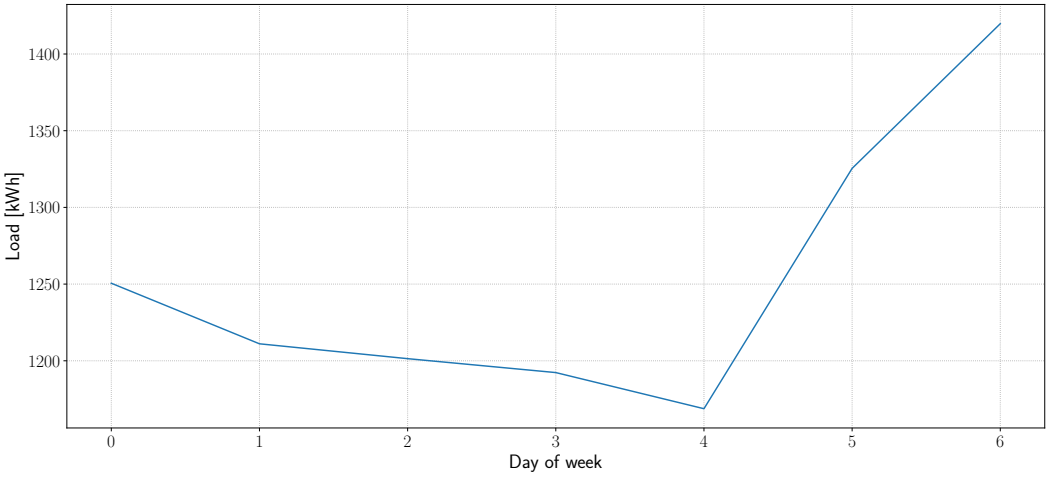

Figure 3.10: Controller Total Day of Week Load

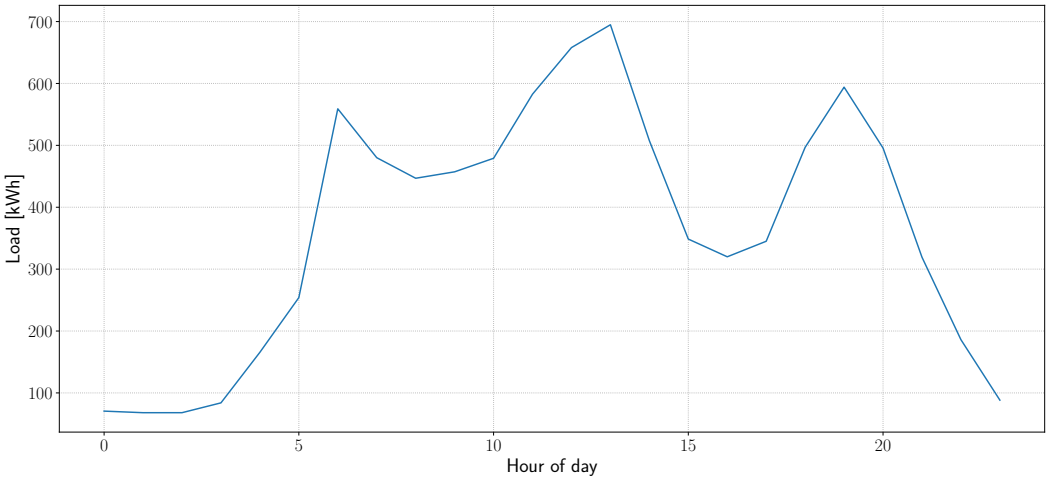

Figure 3.11: Controller Hour of Day Load

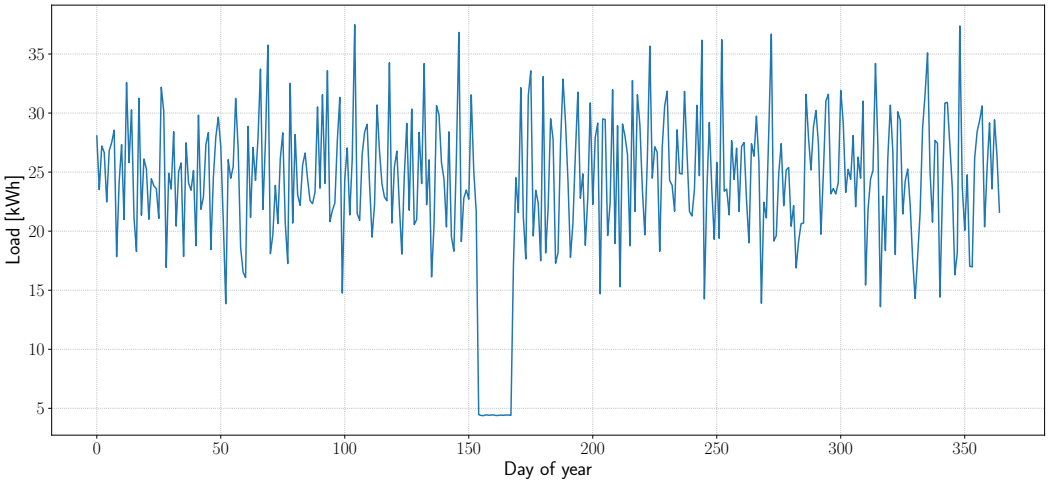

Figure 3.12: Controller Total Daily Load

In Figure 3.9, there is a clear drop in month 5 (June), as the calendar month starts from month 0 to 11. There is a higher use of energy over the weekend, as seen in Figure 3.10. When analysing the 24 hours in Figure 3.11, there are essentially three significant peaks: morning, noon and early evening. There is also a considerable lower energy using during the late night to early mornings, which will attribute to the family's sleeping patterns. For the year, in Figure 3.12, the load profile consistently rises and falls, which can be attributed to the higher energy use during weekends.

3.3.1.2 Weather Profile

The solar profile is for the same period as the load profile, during 2018. Figure 3.13 shows the average monthly irradiance, which is very similar to Figure 3.6.

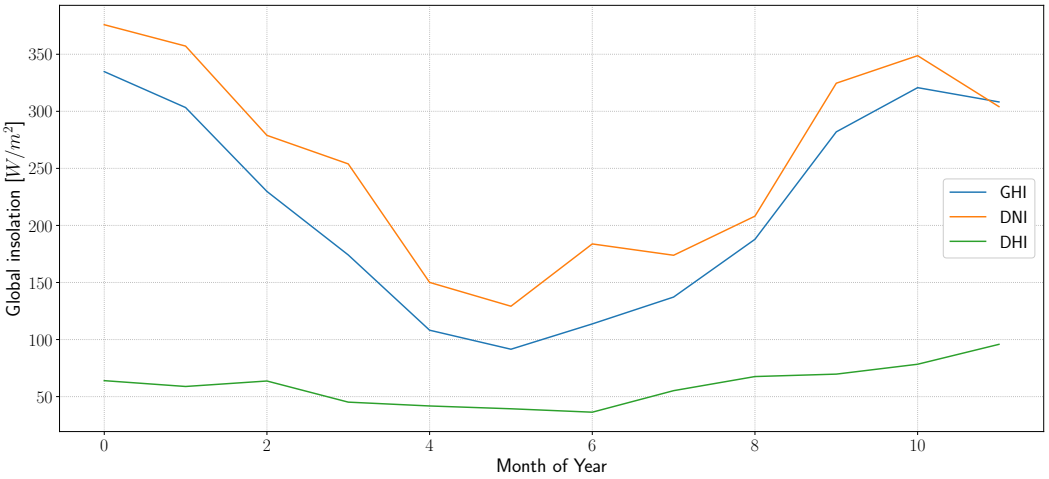

Figure 3.13: Controller Monthly Average Irradiance

Table 3.6: Controller Weather Statistics

Solar component	DNI (W/m^2)	GHI (W/m^2)	DHI (W/m^2)
Mean (μ)	435.2	365.2	101.1
Maximum	1062.3	1098.94	669.0
Standard deviation (ρ_X)	383.7	323.83	104.3

Table 3.6 shows a summary of the average, maximum and standard deviation of the different solar components, namely DNI, GHI and DHI.

3.3.2 Power Source Mathematical Models

The mathematical models used in the controller are similar to that of the GA. The HPS will be a combination of PV-modules, a limited grid connection, back-up battery storage and a diesel generator. The results obtained will be discussed in Section 4.2 of Chapter 4. The sizing of the HPS is done using the GA described in Section 3.1. The sizing results will be based only on the technical aspects as to decide the HPS sizing.

3.3.3 Reinforcement Learning Experimental Design

As discussed in Section 2.5, the RL-based controller consists of an action set, states, rewards and is done using machine learning (ML) with a neural network (NN) architecture. The information is stored in the RL-based controller's memory, from which the NN is trained and improved.

3.3.3.1 Actions

The actions consists of a combination of executable actions. The following actions can be executed:

- Use the PV generated power for the load and/or charge the batteries
- Charge the batteries
- Discharge the batteries
- Use the grid connection for the load and/or charge the batteries
- Use the generator power for the load and/or charge the batteries

Except for the battery charge and discharge actions, all actions can be executed simultaneously. The possible combinations of the 5 executable power actions are thus 32 (2^5) of which 8 is invalid due to the exception of the charging and discharging cannot be done simultaneously. This leaves a total of 24 possible executable action combinations. The action consists of an array of true/false statements and is given by:

$$\begin{bmatrix} \text{Use PV power} \\ \text{Charge batteries} \\ \text{Discharge batteries} \\ \text{Use grid power} \\ \text{Generator state (on/off)} \end{bmatrix}^{-1} \tag{3.3.1}$$

A possible combination of TTFTF is thus interpreted as true (T) for the use of PV power, charge the batteries, not discharging the batteries thus false (F), using the grid power and not using the generator, which will mean to switch off the generator if it was running. It should be noted that the Use-PV-action can be selected regardless of when there is PV available. In the instance where the available PV is zero and the Use-PV-action is selected, it will result in a zero usage of PV power. However, the agent can be rewarded for learning to use PV only during the day, when PV is available, and to use alternatives when PV is unavailable.

3.3.3.2 States

The states consist of the load, available PV power, battery state of charge, month, the hour of the day, whether the day is a weekday or a weekend day and the generator state, being on or off. The number of months varies from 1 to 12 and the hour of the day can be 0 to 23. The weekday is presented as a 1 when it is a weekday and 0 when it is a weekend day. The generator state

represents the number of running hours, where the 0 state refers to an off state. The load, available PV, month, the hour of the day and weekday states are fixed, whereas the battery and generator state is dynamically changed based on the actions in the previous state.

3.3.3.3 Rewards

The rewarding and penalty system is how the RL controller learns. The agent executes an action and receives a reward after the action is completed for the time step. The greatest reward is when there is no power supply loss and thus the entire load is met using the HPS for that time step with a numerical reward of +20. The agent is encouraged to use alternative power options when there is no PV available and is rewarded +3 when it uses the generator or +6 discharges the battery. This will aid in the control system during the months with lower solar insolation where the batteries cannot be fully charged and the HPS relies on the generator and grid. To encourage the controller to use the available PV, it is rewarded with +6 when the PV usage is greater than zero in order to maximise the renewable energy source. It is also rewarded +6 when the total available PV is used for the load and/or charge the batteries.

3.3.3.4 Neural Network Architecture

The RL implementation is done using Machine Learning (ML), which will be done using python and Keras [47, 48]. Keras is a high-level API of TensorFlow for building and training deep learning models and is considered user-friendly, modular and easy to extend. These models use artificial neural networks (ANN). A simplified neural network architecture is shown in Figure 3.14. The principle behind a NN is to imitate the way interconnected brain cells recognise patterns and relationships. The NN has artificial neurons, called units. There are input neurons, hidden neurons and output neurons. A fully connected NN is where every hidden and output neuron is connected to each unit in the layers inside. The relationships between each of these units have a weight.

There is no generic method of designing a NN and very little guidance is given to determine the number of neurons and the number of layers of a NN. The NN is capable of mapping non-linear relationships between input variables. Previous research can be used as a guideline and overall guidelines suggest to start with a small network and then see which alterations improve the network. The optimisation of the hyperparameters has to be done iteratively as well.

The input layer consists of a fully connected layer with the number of states as the input. The activation function is a rectified linear unit (ReLu). The dimensionality of the output space is 32 units. The activation function transforms the summed weighted input into the output activation of that input [49]. The

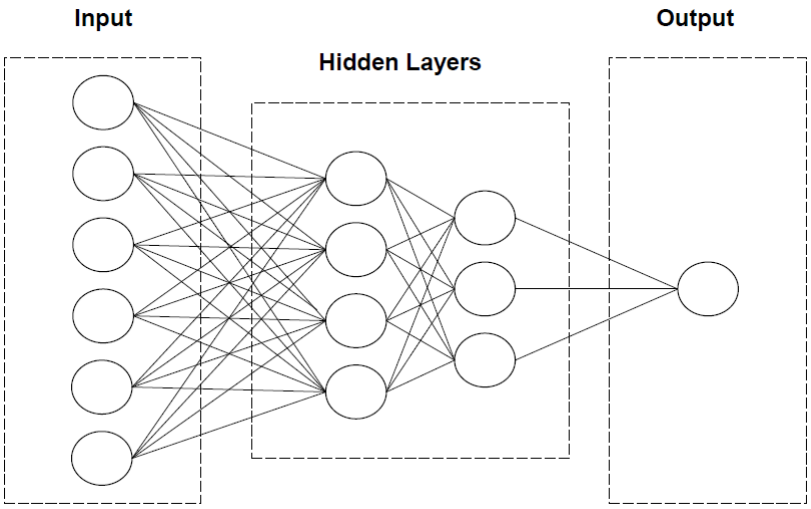

Figure 3.14: Simplified Neural Network

non-linearity allows for complex relationships within the data to be learned. The non-linearity allows for more sensitivity to the activation sum input and circumvents easy saturation [49]. The ReLu function is given by:

$$f(x_i) = \max(o, x_i), \tag{3.3.2}$$

where x_i is the input value.

A fully connected layer, also known as a dense layer, represents a matrix vector multiplication. The matrix values are trainable parameters which are updated during training. The dense layer changes the dimensions of the vector by applying rotation, scaling and translation. The dense layer thus transforms the vector. The hidden layers consist of two fully connected layers with 32 units, with an activation function ReLu. The output layer has the number of actions as the output which is 24 possible actions. The output represents the Q-value of each action. The Q-value is the maximum future discounted reward the agent can obtain if it executes an action in the specific state and is shown in (2.5.11).

The loss function is the mean square error (MSE) between the Q-values and the estimator. The optimiser is the Adam optimiser which will minimise the loss function. Adam is an extension to the stochastic gradient descent [50] and the stochastic gradient descent maintains a single learning rate for all the weight updates and thus the learning rate remains unchanged. However, for Adam optimiser, the learning rate is maintained for each network weight and is separately adapted as the learning unfolds. Adam realises the benefits of adaptive gradient algorithm and root mean square propagation and can thus handle sparse gradients on noisy problems and is relatively easy to configure.

The hyper parameters are specified as the following:

1. Learning rate α: 0.001,
2. Start ϵ: 1,
3. End ϵ: 0.01 ,
4. Discount factor γ: 0.7,
5. Episodes: 50,
6. Batch size for optimiser: 3 hours.

The start and end ϵ refers to the use of the ϵ-greedy policy, as explained in Section 2.5. As each episode continues, the ϵ-value starts at 1 and decays to 0.01, which aids the exploration and exploitation in the algorithm. In (2.5.11), the discount factor delays the reward's impact. A higher discount rating means that the algorithm is long term bias, whereas a value closer to zero has a short term bias. The learning rate α updates and normalises the current state. These values were based on literature and iteratively analysing the results from the different NN architectures. The number of episodes refers to how many episodes the controller has to learn the system. With each episode, the epsilon value decreases which allows for more exploitation of previous knowledge.

3.3.3.5 Memory

The memory is updated after each time step and the RL controller chooses random 3-hour batches to retrain the network after each time step. It was decided to use random selections as the RL controller does not need a sequential list of states, but rather its previous state, the action performed, the next state and the reward obtained. The memory reduces the risk of over-fitting the network. In addition, the size of the memory has to be large enough to ensure enough information is stored, but not too excessive that it wastes computational power.

An analysis of what intervals are deemed appropriate for the controller has to be done. Training the network after every minute interval is computationally expensive and unnecessary. However, large intervals might lead to inaccurate representations of the load and the controller's input. Thus a trade-off between computational usage and accuracy takes place. An analysis of different training of 5, 10 and 15 minute intervals is done. As the interval frequency increases, the training time of the NN increases accordingly. It would be inefficient to have a real-time controller which trains too long and causes a bottleneck of data in the hardware. As the intervals increase in frequency, the amount of data required increases significantly. The use of hardware memory of GPUs and CPUs is limited and thus alterations to different memory sizes have to be considered. This interval analysis was done based on the entire database with

a simplified NN architecture and reward system. The purpose of this analysis was to determine whether it is necessary to use a higher frequency when a lower frequency will give the same results. Figures 3.15 and 3.16 show how the RL controller converges to an optimal point.

The total rewards per episode are normalised to give an accurate representation of how the different intervals perform. Both the rewards and LPS converge, indicating that the agent has learnt optimal policies as part of the control system. The three intervals all show promising results. However, the training time varies significantly. The training difference between the 15-minute and 5-minute intervals varies more than 24 hours. A lower frequency shows a possible convergence with a significantly reduced training time. It would be impractical to execute an action every 5 minutes and the training time causes

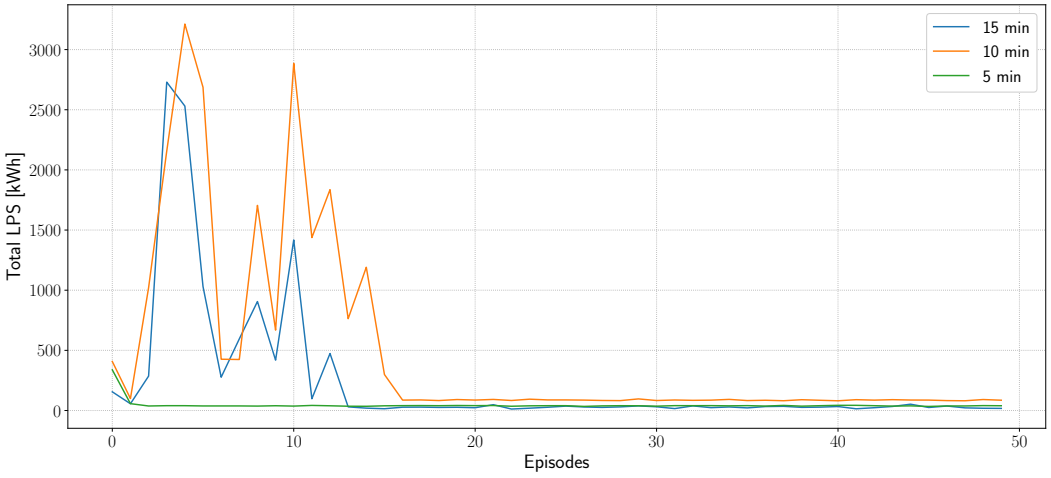

Figure 3.15: Analysis of Controller Time Step Intervals: LPS

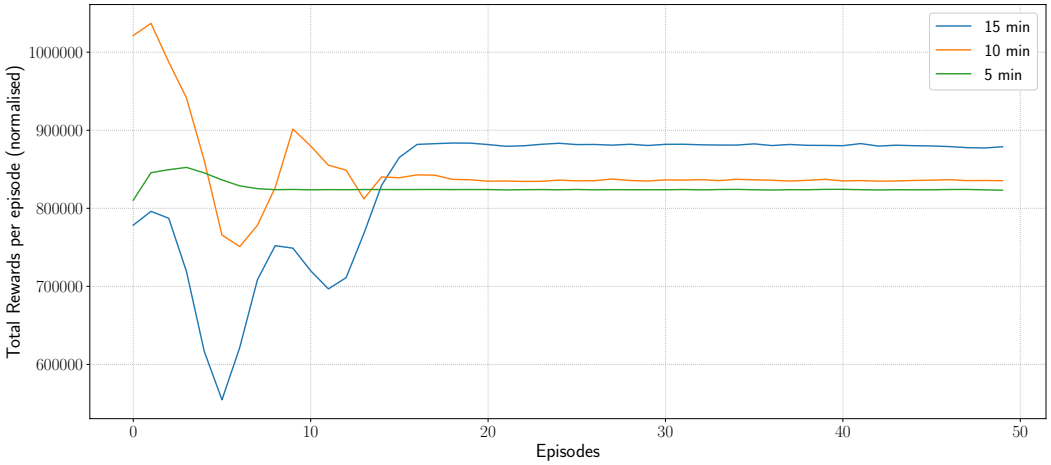

Figure 3.16: Analysis of Controller Time Step Intervals: Rewards

a bottleneck of data processing. Higher frequencies can be more accurate, but the hardware limitations have to be considered.

For this control system, it was decided to use a 15-minute interval for simulation of the RL-based controller of an HPS. The memory points for the 15-minute interval training frequency is selected as two times the length of the training database. The NN trains batches from this memory and reduces over-fitting the architecture.

3.3.4 Evaluation

The evaluation of the control system is based on two factors: the minimisation of the LPS and the maximisation of the PV power. To consider the cost optimisation as part of the analysis, the reduction in generator use and grid use will also be used as a measure of evaluation. Monetary value in cost optimisation is not done as the simulations become computationally expensive and the hardware limitations associated with the available memory to run these calculations. Instead, a consideration of the PV usage against the generator is considered. The PV and battery installations have large capital costs, but a lower operational cost than the generator. A system which utilises the energy exchange of the PV and battery more than the use of a generator is thus seen as a more cost-effective control system.

The data is split into three sets: training, validation and testing. Because of the cyclical nature of the load and weather profile, each month is split into three sections. The ratio of the training-validation-testing is specified as follows

- Training: Days 1 to 22 of the month, which account for 72.3% of the total data,
- Validation: Days 23 to 26 (13.2%),
- Testing: Days 27 to month end (14.5%).

In this way, each month is part of the training, validation and test set. In doing so, each of the different seasons can be trained in the NN.

3.4 RL-based Control System with IoT

The energy exchange between the different components can be increased if the RL-controller has prior knowledge of what to expect with regards to solar power. A dataset of predictions has to be developed from scratch, as no database is available for this.

A proof of concept with regards to predicting solar irradiance using weather predictions has to be done to show that it can be an accurate source of information. This is done using a linear regression algorithm. The NN has to be redesigned to consider the additional information and to assess if this is useful for the RL-based controller. The purpose of this section is to show that a linear relationship between the weather predictions and expected solar irradiance does exist. The RL-based controller's NN will be able to learn this relationship during its training process. However, if no relationship occurs, the data will not benefit the RL-based controller.

3.4.1 Analysis of Solar Irradiance Predictions Using yr.no API

3.4.1.1 Data Scraping Using yr.no API

In order to simulate as though the controller is pinging the yr.no website hourly, a database of these scraping data had to be generated. Since there is no database with the predictions on the hour from yr.no, the database had to be generated from scratch. The data was recorded hourly and this data will be used as part of the control system. Gaps in the database occurred as a result of load shedding and network problems. The analysis of the data is discussed in Section B.2 of Appendix B. The scraping programme code can be seen in Section C.2 of Appendix C.

It should be noted that these values consider night time predictions, where the actual irradiance will be non-existent, which can skew the data. However, for the purpose of showing that there are relationships between the predicted and actual values, it is sufficient. A timing component, such as the hour of the day and month, can increase the accuracy of the predictions.

In addition, yr.no only updates the predictions every 12 hours, which can influence the accuracy of the results. Due to the scraping of the website, there are gaps in the data where load shedding turned off the computer accessing the website and other network errors occurred. Due to the nature of the updating frequency of yr.no, the missing data can be filled by the closet ping to that time. However, large periods of missing data cannot be filled and will thus be discarded, resulting in a smaller usable database. However, to increase the amount of data, this problem can be solved by splitting the data into sequential batches, rather than one large database to validate the system.

3.4.1.2 Preliminary Results of Linear Regression

The yr.no website is accessed through the API to extract the current weather predictions of a certain location. The implementation code can be seen in Section C.3 of Appendix C. The co-ordinates and altitude of Stellenbosch

were used as the location of the controller due to the availability of data. A linear regression using a neural network is used to extrapolate the expected irradiance from predicted yr.no data. The data was split into an 80/20 training and test set. The test is part of the evaluation to validate the results.

The NN has the following inputs:

- Timing components month and hour values
- Temperature and wind speed predictions
- Overall clouds, low-, medium- and high clouds
- Solar components using the clear sky global irradiance, which is calculated based on the time. The calculations are discussed in Section A.1 of Appendix A.

The information is normalised using a standard normal distribution for a z-score and is given by (3.4.1). The goal of normalisation is to change the values of the data to a common scale. This scaling does not distort the variances and does not lose vital information. The normalisation allows for the NN to model the data correctly.

$$z = \frac{x - \mu(x)}{\sigma(x)} \tag{3.4.1}$$

The linear regression model is a fully-connected sequential NN and the input layer has an output of 32 neurons. The activation function is ReLu. The hidden layer has 16 output neurons with a ReLu activation. The output layer has one output, namely the actual irradiance. The optimiser is RMSprop, a gradient-based optimisation technique, and the loss function is quantified by the mean absolute error (MAE).

The model is trained using the training set and the model then predicts by using the test set. The model predicts the expected global irradiance and then compares it with the actual global irradiance. The MAE between the expected and actual irradiance is the loss function which has to be minimised, which will be updating the NN during the training of the model. The results are presented in Figures 3.17 and 3.18.

In Figure 3.18, the expected results and the actual global irradiance seem to have a linear pattern, as expected from a linear regression algorithm. The deviation in error is shown in Figure 3.17 and shows the swing of error. The MAE is 50.46 W/m^2. It shows a high count of small errors but still has a large error swing.

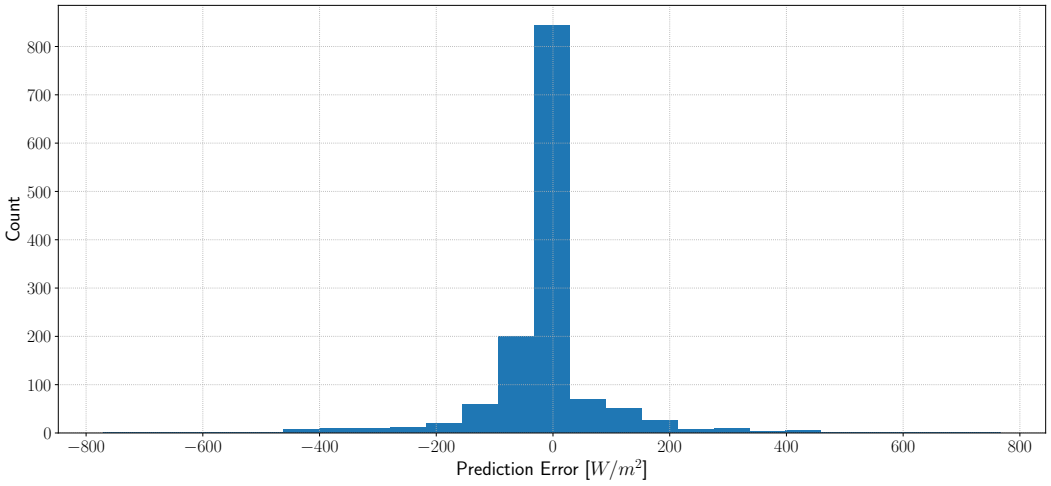

Figure 3.17: Prediction error

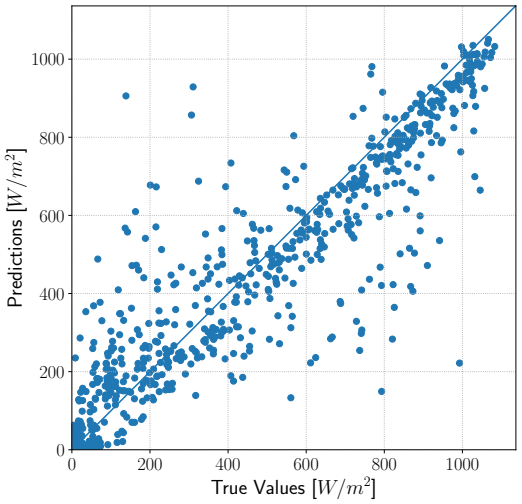

Figure 3.18: True vs Predicted values

When the previous hour's irradiance is also used as an input for the NN, the accuracy of the predictions are increased and the results can be seen in Figures 3.20 and 3.19. The MAE is reduced to 33.4 W/m^2, an improvement of 33%. Comparing Figures 3.18 and 3.20, the linear regression with prior knowledge has a less scattered linear graph. Comparing the error swing in Figures 3.17 and 3.19, the prior knowledge linear regression has a smaller error swing.

The controller cannot predict the immediate future, but taking

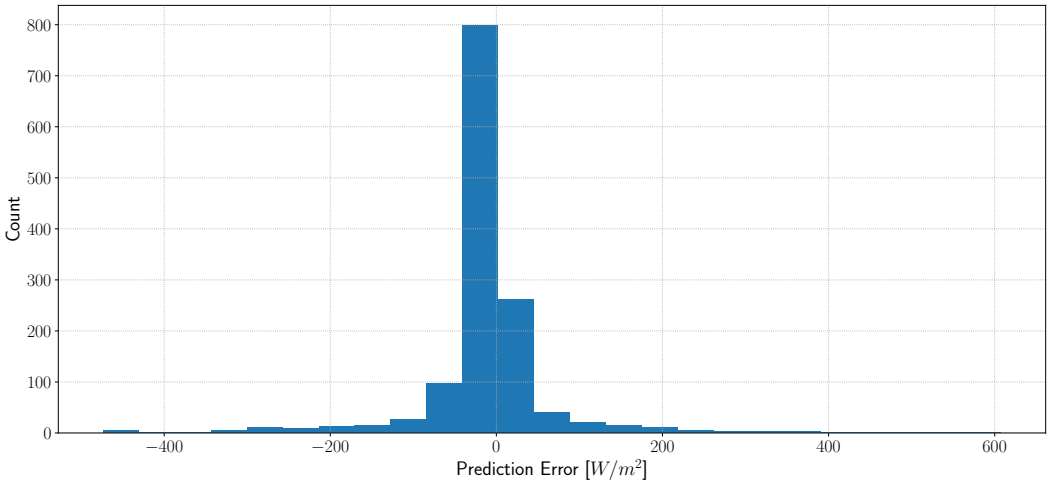

Figure 3.19: Prediction error with prior knowledge

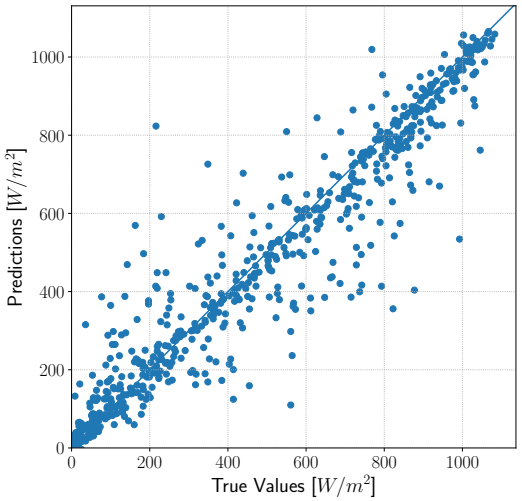

Figure 3.20: True vs Predicted values with prior knowledge

3.4.2 Reinforcement Learning with IoT Experimental Design

3.4.2.1 Actions

The set of actions are identical to those discussed in Section 3.3.3.1.

3.4.2.2 States

The same states are applied as the RL-based controller, as discussed in Section 3.3.3.2. Additional environmental information scraped from the yr.no website is fed into the NN. The information is scraped on the hour and used for the entire period and can be seen in Table 3.7. The weather predictions are nor-

Table 3.7: Attributes obtained from yr.no

Attribute	Unit
Clouds	%
Low clouds	%
Medium clouds	%
High clouds	%
Temperature	°C
Humidity	%

malised using the z-score. The clear sky insolation of the previous, current and the next time step is also added to the states.

3.4.2.3 Neural Network Architecture

The hyperparameters are identical to that discussed in Section 3.3.3.4. The optimiser and loss function is also the same. The input layer has the original states, as well as the prediction information as given in Table 3.7. The additional information did not increase the size of the NN. The input layer has 32 neurons and is activated using ReLu. The two hidden layers have 32 neurons with an activation function ReLu. The output layers are still the estimated Q-values of different actions.

3.4.2.4 Rewards

The rewards are identical to that discussed in Section 3.3.3.3.

3.4.2.5 Memory

The memory size will also be identical to that discussed in 3.3.3.5.

3.4.3 Evaluation

The evaluation of the RL-based controller with IoT will be based on the same evaluation as discussed in Section 3.3.4.

3.5 Baseline controllers

Two baseline controllers were used to assess the validity of the RL-based controller. The random action controller implemented random actions, whereas the rule-based controller was based on predefined rules.

3.5.1 Random Action Baseline

The workings of a random-action-based control system can be seen in Algorithm 2. The action A is randomly selected and is then implemented for that specific period. The controller does not observe any of the environments and directly influences the next state.

Algorithm 2 Random Action Control System

Input: $\{s, a\} \in \{\mathcal{S}, \mathcal{A}\}$

Loop for each episode
  Loop for each time step
    Choose A randomly
    Observe the immediate next state S'
    $S \leftarrow S'$
  Until S is terminal

Output: LPS, Generator usage, PV usage, Battery usage, Grid usage and rewards for each episode

3.5.2 Rule-based Action Baseline

Algorithm 3 shows the workings of the rule-based controller. An action array, as shown in (3.3.1), is generated based on rules. Each power source will receive a True or False setting for it to be used (or not used). These values then create an action array of T or F values, as explained in Section 3.3.3.1. The rules are as follows:

1. If PV is available: use PV,
2. If PV is greater than Load: charge batteries;
3. If PV is less than load: discharge batteries;
4. Randomly select the use of the grid;
5. Randomly select the use of the generator.

The controller observes only the immediate state, creates an action based on the rules and that action influences the next state. The grid and generator use are randomised. Because the action array is sequentially formulated, if the grid was always selected, it influenced the use of the generator. The cost optimisation objective is to reduce generator and grid usage, but not to exclude any of them.

Algorithm 3 Rule-Based Action Control System

Input: $\{s, a\} \in \{\mathcal{S}, \mathcal{A}\}$

Loop for each episode
  Loop for each time step
    Choose A based on rules
    Observe the immediate next state S'
    $S \leftarrow S'$
  Until S is terminal

Output: LPS, Generator usage, PV usage, Battery usage, Grid usage and rewards for each episode

3.6 Conclusion

An experimental design to test whether a GA is a viable design option for an HPS is presented. The algorithm assesses various aspects, being technical, financial and environmental, to optimally design an HPS. HOMER is used as a comparison tool to highlight the similarities and differences between the two design methods and will be discussed in Section 4.1 of Chapter 4.

A controller using a model-free Q-learning RL is designed to assess its validity as a control system. Additional information is extracted from yr.no to aid in predicting PV power and this is incorporated in a separate RL-based controller. Two baselines are used to compare the results and will be discussed in Section 4.2 of Chapter 4.

Chapter 4

Results

4.1 Design of a Hybrid Power Supply

This section reviews the results of using a GA as a design tool for an HPS. In the previous chapter, the experimental design to implement this algorithm was discussed and highlighted the mathematical models used, as well as any other relevant assumptions. The design attempts to optimise three main objectives: technical, financial and environmental. The technical objectives aim to reduce the loss of power supply, whereas the financial objectives are focused to provide the investor with the maximum theoretical ROI. The environmental objectives aim to reduce the use of the generator and to maximise the use of the RES. The GA was modelled using Python 3 and the programming code can be viewed in Section C.1 of Appendix C. The load and weather profile was simulated in HOMER and the results will also be discussed below. The assumptions for HOMER was discussed in the previous chapter and were tried to be as similar to the GA's assumptions as possible. A discussion between the two design tools is presented to highlight the similarities and differences between them.

4.1.1 Genetic Algorithm

4.1.1.1 Configurations

The top configuration results are shown in Table 4.1 and the components can be viewed in Table 4.2. The different configurations are Solar-Grid (SG), Solar-Grid-Battery (SGB), Solar-Grid-Generator (SGG) and Solar-Grid-Battery-Generator (SGBG). The LPSP and LPSP critical refers to (3.1.9). The capital cost refers to the initial input cost to the investor and the ROI refers to the annual return on investment for the investor. The PV utilisation and Generator utilisation refers to (3.1.14) and (3.1.15) respectively.

The SGBG has the greatest solar power utilisation, which accounts for the shifting of solar power to the load and back-up storage for the HPS. For the

Table 4.1: Genetic Algorithm results of different HPS configurations

Configuration	LPSP (%)	LPSP Critical (%)	Capital Cost (ZAR)	ROI (%)	PV Utilisation (%)	Gen Utilisation (%)
SG	42.1	26.9	R3,051,177	23.4	47.5	0.0
SGB	23.9	13.4	R5,335,826	26.2	64.5	0.0
SGG	8.3	5.3	R2,405,780	19	40.0	40.9
SGBG	0.01	0.01	R4,400,192	18.3	91.1	37.6

SG configuration, there is a considerable difference in the critical vs non-critical percentage, compared to the other configurations. The investor can use this insight to view the trade-off between when and how much the loss of power supply occurs. This insight can influence changing certain production hours to change to load profile, which in turn can take full advantage of the solar power production. The SG configuration has the highest LPSP, which is to be expected as there is no back-up storage or additional generation during non-daylight hours, except for the limited grid supply.

The addition of batteries to the configuration increases the capital costs considerably because of the higher expense. Optimising battery usage is vital in an HPS because this can increase the life expectancy of the plant, which in turn will reduce replacement costs. The return on investment of the SGB configuration was the highest and also the top user of PV power. The SGG configuration has a low LPSP with a considerably lower capital cost than the SGB configuration. However, the use of fuel increases the operational and maintenance costs, which lowers the ROI for the investor. As with the batteries, the operational and maintenance costs are vital in maximising the life expectancy of the HPS.

As more power sources are incorporated, the loss of power supply becomes close to zero. This is due to the back-up storage of the batteries and the additional generation of the diesel generators. Though SGBG configuration has the lowest ROI, this HPS ensures that the entire load is almost always met. The ROI of the different configurations show promising results for the investor. The analysis does not include any tax benefits/deductions for the investor, which will influence the results. The excess energy generated by the solar power is not fed back into the grid. Such an addition can increase the investor's ROI by creating an additional income source for the HPS.

Concerns can be raised when incorporating batteries and generators. With a large focus on reducing pollution and its effects, concerns about the disposal of batteries after their lifespan have been raised and the emissions of diesel generators have been raised. The sustainability of fuel can become a threat to the HPS as the scarcity of fuel and thus the price of fuel, will increase. Future

Table 4.2: Genetic Algorithm results of components in different HPS configurations

Configuration	Name	QTY	Power
SG	Canadian Solar 395W	551	217.6 kW Peak
	Sunny Tripower 15000TL	29	
SGB	Canadian Solar 395W	684	270.2 kW Peak
	Sunny Tripower 25000TL	36	
	OmniPower 240Ah 12V	190	Storage 127.6 kWh
SGG	Canadian Solar 395W	323	127.6 kW Peak
	Sunny Tripower 20000TL	19	
	MAC AFRIC 34 kVA	3	102.0 kVA
SGBG	Canadian Solar 365W	312	113.9 kW Peak
	Sunny Tripower 25000TL	26	
	MAC AFRIC 50 kVA	3	150.0 kVA
	OmniPower 180Ah 12V	190	Storage 410.4 kWh

carbon tax laws may also be implemented, which further increases uncertainty over the lifetime feasibility of the HPS. The components for the different HPS configurations are shown in Table 4.2. As the technology of PV-modules, generators, inverters and batteries increases, the systematic replacements of these components will improve with better performing power generation and greater efficiencies. The O&M of the HPS is vital in ensuring the maximum use and profitability of the plant.

4.1.1.2 Loss of Power Supply

The load and weather profile of the farm in the Stellenbosch area is briefly analysed and discussed in Sections 3.1.1.1 and 3.1.1.2 respectively. The analysis showed a higher load at the beginning of the calendar year. The climate of the specified location has a higher PV yield at the beginning of the year and at the end of the year, where a lower yield is expected during the winter months of April to August. The load profile has a spike at the beginning of the month and the average hourly usage peaks from 09:00 to 16:00. The hourly LPS of the different SG, SGB and SGG configurations can be seen in Figures 4.1, 4.2, 4.3 and 4.4.

Considering the SG's LPSP of 42.1% and Figure 4.1, it is clear that LPS occurs often. There are also clear spikes which correlate to the higher spike in the load profile. The battery storage reduces the LPS considerably and the higher spikes are attributed to the change in load profile at the beginning of the month, as seen in Figure 4.2.

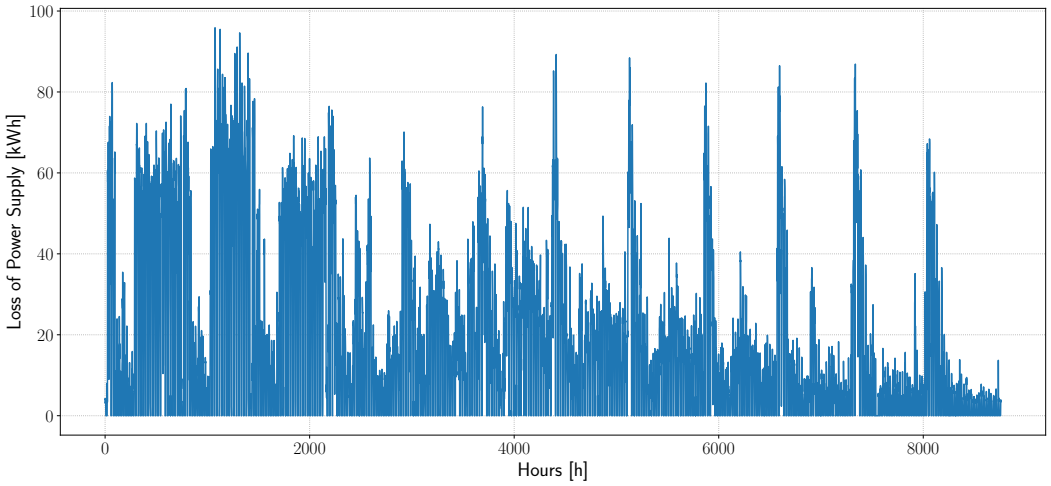

Figure 4.1: Hourly loss of power supply from SG configuration over 1 year

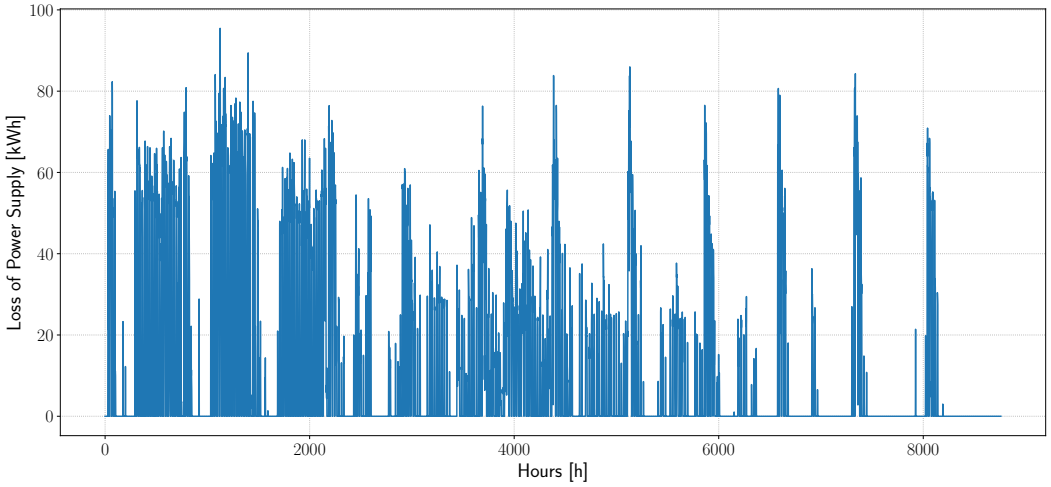

Figure 4.2: Hourly loss of power supply from SGB configuration over 1 year

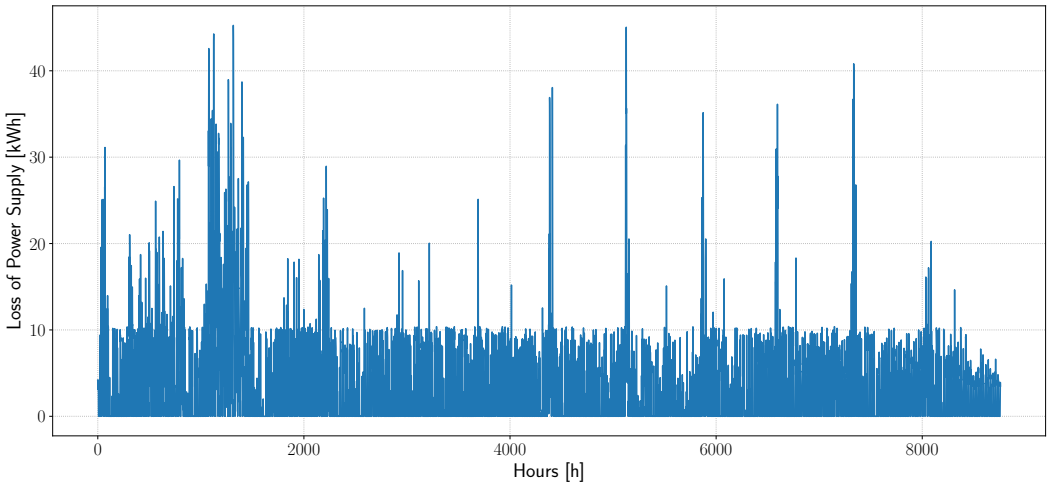

Figure 4.3: Hourly loss of power supply from SGG configuration over 1 year

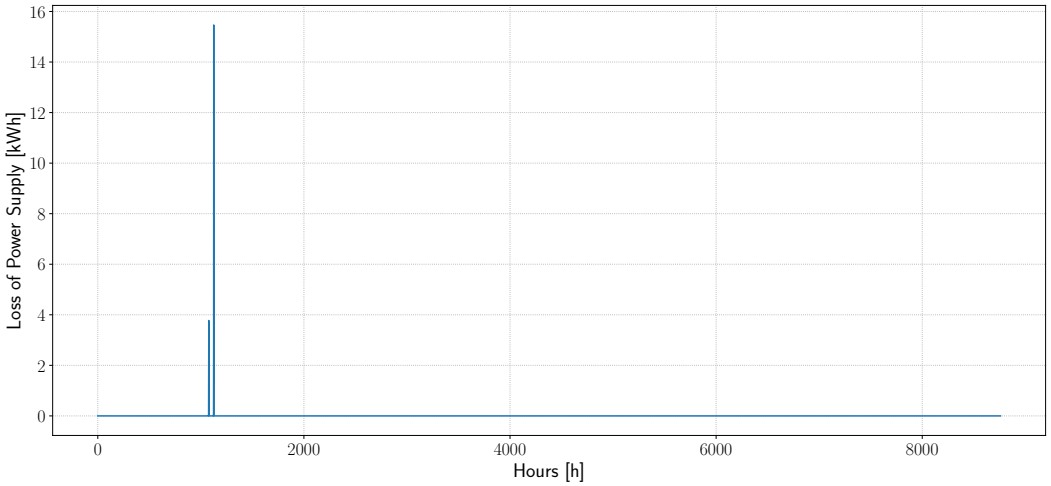

Figure 4.4: Hourly loss of power supply from SGBG configuration over 1 year

In Figure 4.3, the configuration has regular LPS, but it is considerably lower than the SG configuration. The SGB configuration has a lower occurrence of LPS, but the hourly LPS is considerably lower for those few occurrences. Increasing the search space of the number of generators or increasing the grid connection limit will decrease the LPS of the SGG. In terms of the technical objectives, the SGBG is the configuration which meets the criteria the best. In Figure 4.4, the only spike occurs at the one peak of the load profile. The SGBG configuration has a 0.01% LPSP and utilises the PV power the most. It is also important to note that the PV capacity size is considerably less than the SG and SGB, which may influence the utilisation factor. The battery storage is also less than the SGB configuration.

There is a cost trade-off to decrease the LPSP. The investor has to consider how much more expensive it will be to decrease the percentage of the LPSP. As the percentage becomes smaller, the cost increases considerably. The simulation accounts for a simple rule-based control system, which may influence the system's performance. Creating a controller which can study the load profile might give a better insight into how the control system must be designed.

4.1.2 Design using HOMER software

The assumptions made in Chapter 3 in Section 3.2 were simulated using HOMER software [18]. The results obtained can be seen in Tables 4.3 and 4.4. In HOMER, the ROI is calculated based on a reference system and not on the system's actual performance and is therefore excluded from the results [51].

The SG configuration has the lowest capital costs and annual O&M costs, however, the capacity shortage is considerably higher. The replacement costs are also the lowest as the PV array has a longer lifetime usage. This is where the

Table 4.3: HOMER results of components in different HPS Configurations

	PV Array (kW Peak)	Diesel Generator (kW)	Battery (kWh)	Grid (kW)	Converter (kW)
SG	233	-	-	10	26.4
SGB	44.6	-	190	10	13.7
SGG	400	89	-	10	52.1
SGBG	89.8	89	280	10	57

Table 4.4: HOMER results of different HPS configurations

	Capital (ZAR)	Replacement (ZAR)	Annual O&M (ZAR)	Annual fuel (ZAR)	Capacity shortage (%)	Renewable fraction (%)
SG	R1,668,076	R241,771	R163,587	R0	50.1	64.6
SGB	R2,859,711	R2,333,100	R184,560	R0	24.8	65.4
SGG	R3,326,393	R1,937,539	R249,507	R1,710,822	0	34.3
SGBG	R3,483,711	R2,728,488	R181,302	R185,727	0	71.1

trade-off between the different objectives come into play again. The investor's initial costs and operational costs will be the lowest, however, the possibility to have a loss of power supply is considerably higher.

The SGB has the smallest PV capacity. Because the grid's cost of energy is the highest, HOMER tries to optimise the cost of energy by using the grid as an energy source, rather than the PV. The battery will charge and discharge often and this will decrease its life cycle usage. This is also evident in Table 4.4 with the replacement costs. The capacity shortage is 24.8% which is very similar to the GA's LPSP.

The SGG and SGBG's capital costs are very similar where the SGBG has a slightly higher capital cost. However, the annual O&M and fuel costs are considerably lower for the SGBG. The additional battery storage decreases the amount of fuel use significantly. The SGBG has the highest renewable fraction percentage, but it is important to note the smaller PV array size. Both the SGG and SGBG have a zero capacity shortage. These two configurations rely on the generators to add additional power when the PV, grid and battery (SGBG) cannot fully supply the load.

4.1.3 Discussion

The design using a GA was done to optimise the HPS size. The ultimate goal is to design an HPS configuration which minimises the LPS, while still maximising the use of renewable energy and minimising the use of fossil fuel sources while giving the investor an acceptable return on investment. The

addition of more power sources reduced the loss of power supply as the different advantages of the power sources are exploited. The weaknesses of one power source are replaced with the strength of another, as seen in the SGB, SGG and SGBG configurations.

Comparing the results of HOMER and the GA, the HOMER SGG configuration had a lower capital cost, whereas the GA's SG, SGB and SGBG had higher capital costs. Though the capital costs are lower than the GA's, the reliance on fuel is also considerably more. The equipment is replaced more often because of the overuse of components. The GA, because it can be modified, assesses each battery and generator individually and replaces it according to its usage. The capacity shortage of the two generator configurations is zero because they rely heavily on fuel to supply the load, which is not the objective of the design process.

HOMER software cannot override certain objective goals, whereas the GA can, by looking at configurations which has more renewable energy usage and less generator usage. HOMER does not specify how many components are required, but rather a value. HOMER thus does not assess individual components, but rather the overall capacity of the power source. The GA also considers multiple type of components, based not only on the power ratings but also its financial indicators. As an example, a PV module with a higher cost, but a longer warranty may be considered by the GA as a more viable component, compared to one with a lower cost, but a considerably shorter warranty.

There is a difference in the component sizing between the GA and HOMER, which influences the financial aspects of the two designs. However, the GA is constrained to look for options which use fossil fuel-based options as a last resort, rather than the cheapest option. In addition, an HPS which relies heavily on diesel generators runs the risk of being subjected to future carbon tax laws, as well as possible oil shortages or escalating fuel prices. The different control systems implemented in the GA and HOMER may also have an influence on the feasibility of the HPS.

4.2 Control of a Hybrid Power Supply

In the previous chapter, the control of an HPS using RL is discussed. The RL-based controller has no prior knowledge of the system and formulates policies through observation of the environment and trial-and-error search. The agent is given rewards and penalties after it executes an action and uses the Q-learning method in order to maximise the future rewards it can acquire. The control system had a different load profile than the GA and thus a sizing of the HPS was done using the GA. The configuration of the HPS consists of PV-

modules, battery storage, a back-up generator and a limited grid connection. Two baselines are used to compare the results of the RL-based controller to assess the performance of the RL-controller.

The RL controller has two objectives: to minimise the loss of power supply and to optimise the cost of the HPS. The cost optimisation aims to reduce the use of fuel and grid connection which the investor has to fund throughout the course of the year, whereas the energy exchange between the PV and battery storage has a large capital cost, but lower operational costs. An IoT-implementation is also done by adding the weather predictions as additional information for the RL-based controller. These results are compared to the two baselines and the RL-only controller.

4.2.1 Sizing of HPS

The design for the HPS was done using a modified GA which only attributed the technical aspects for 1 year in terms of LPSP. The grid is specified as 1.2 kWh maximum usage per hour with a 100% reliability. The PV capacity is sized as 10.6 kW peak, the battery storage as 49 kWh with a DOD of 50% and has a 5 kW generator.

4.2.2 Reinforcement Learning-based Controller

The training set is used to train the NN, whereas the validation set is the first test of the model. The validation set is used to find suitable NN architecture and optimise the hyper-parameters.

Figure 4.5 shows how the RL agent tries to maximise the total reward. In each episode, the weights of the NN is adjusted and retrained to find the maximum Q-value for each action in the state. The model converges, but slight deviations can be noted. Figure 4.6 shows how the agent tries to minimise the total LPS per episode. The LPS converges to a minimum, which indicates that the agent has started to exploit its previously obtained knowledge. The LPS converges to a minimum and slight deviations can be noted in the figure. In Figure 4.7, the utilisation of the different power sources are highlighted. The PV and battery are considerably higher than the generator and grid, indicating that the agent has learned to utilise the RES and the shift of energy to the storage system.

Equation (2.5.5) in Chapter 2 shows the expected return starting from state s. A discount factor closer to one is more long-term biased, whereas closer to zero is a shorter-term bias. A higher discount parameter significantly favoured the generator and grid usage rather than the PV and battery energy exchange. This is because the highest reward given for a zero LPS. For the long term, the agent could obtain the most rewards by utilising these consistent power

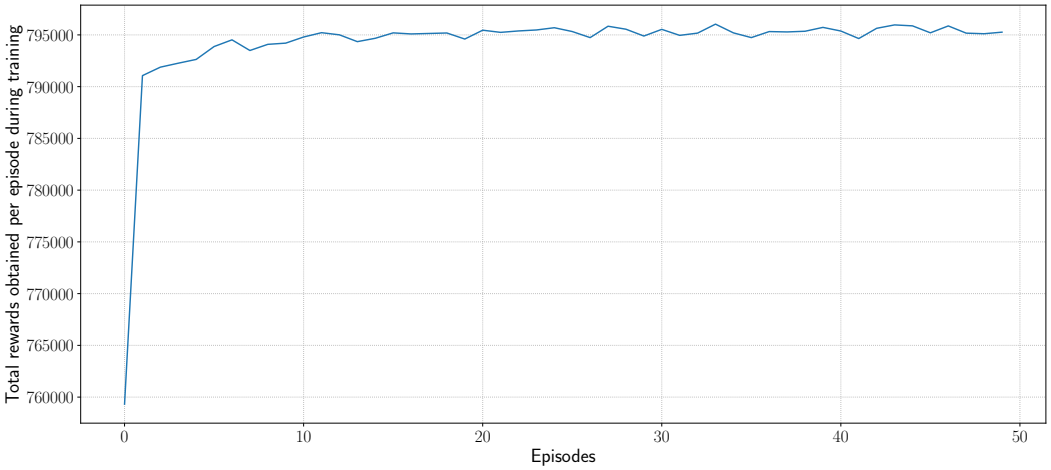

Figure 4.5: Training rewards per episode

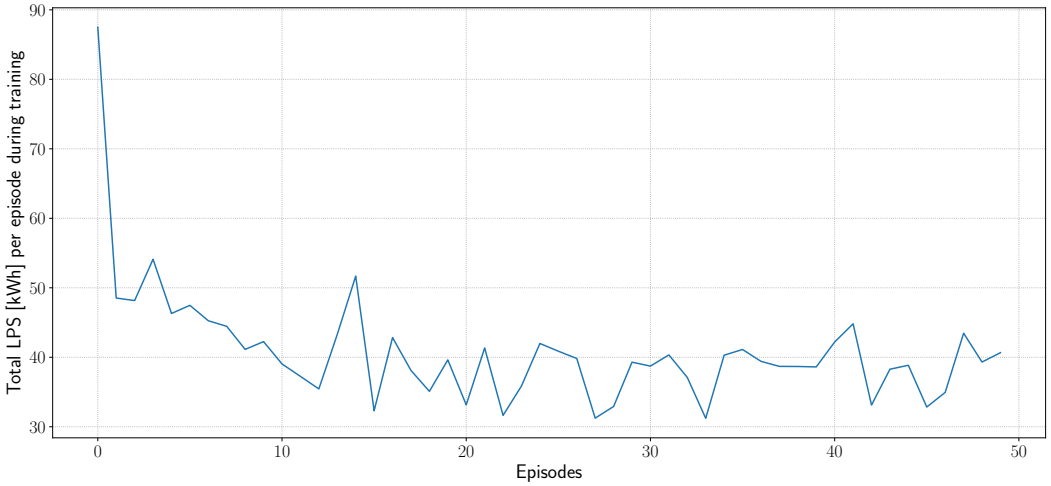

Figure 4.6: Training LPS per episode

sources, such as the grid and generator and this can be observed in Figure 4.8. Initially, the agent explores to find an optimal solution to receive the maximum rewards. The higher generator and grid will result in a long-term no LPS, which gives a much greater future discounted reward and this is exploited clearly in the figure. The agent thus does not struggle to find an optimal solution, because it is heavily dependent on the generator and the grid and does not use the battery to increase the use of the RES. The model minimises the LPS easily. However, the controller relies considerably more on the generator and grid than on the available PV and is seen as cost-ineffective. When Figures 4.7 and 4.8 are compared, the convergence of the higher discount is considerably less than the lower discount factor. The small deviations in Figure 4.7 is an indication that the agent had some difficulty in finding an optimal solution for the HPS.

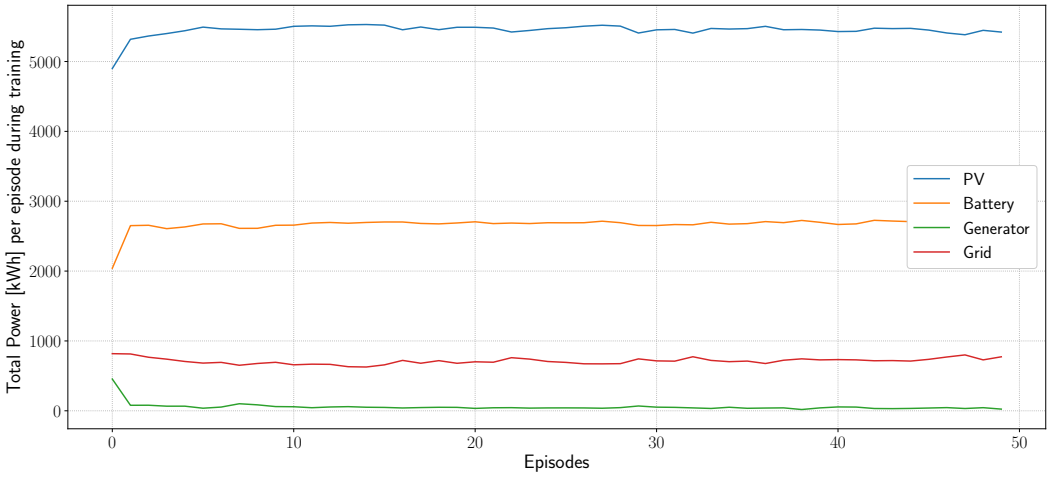

Figure 4.7: Training power source usage per episode

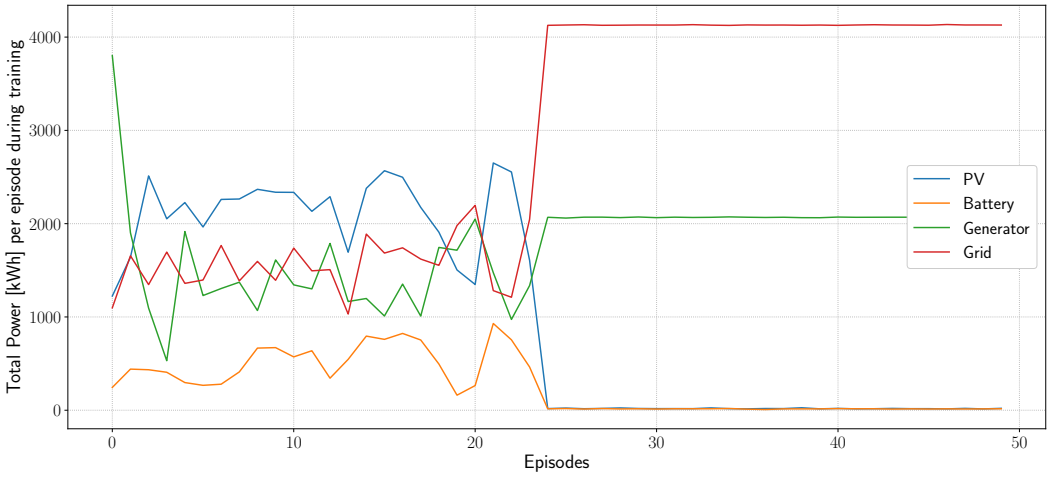

Figure 4.8: Training power source usage per episode with γ=0.995

After training the model using the training set, the model is validated. The validation process constantly iterates over different NN architectures and hyperparameters to find the best-suited model. The most appropriate model's validation results is shown in Table 4.5. The RL-based controller obtained the highest rewards. The rule-based controller has a significantly lower LPS and higher PV usage. The rewards of the rule-based controller is lower, yet the LPS is better than the RL-controller. The RL-based controller learns to use the generator efficiently at night, when the available PV is zero. The rule-based controller does not optimise for this and therefore the RL-controller has a higher reward value. The validation results show that the RL-controller has effectively learned how to control the system from without any prior knowledge of the system, but is not more efficient and optimised with regards to costs. A higher discount factor will utilise the generator and grid more, which

Table 4.5: Validation Baseline Comparison of Control System: RL

	Rewards	LPS [kWh]	Battery Usage [kWh]	PV Usage [kWh]	Generator Usage [kWh]	Grid Usage [kWh]	Battery Charged [kWh]
Random	100231	240.0	300	489.9	176.8	284.9	300.3
Rule	126975	13.1	524.6	1088.5	10.14	77.2	522.0
RL	134807	58.3	477.9	907	38.4	184.3	474.4

can then reduce the LPS of the system, but not optimise the use of the RES. The trade-off between having lower LPS results in a suboptimal cost. Previous models did show a considerably greater reduction in LPS, but the use of the generator and grid was significantly higher.

The total load for the validation set is 1191.6 kWh. Energy balance equation of the system is given by

$$\text{Load} + \text{Battery}_{charge+losses} \approx \text{Bat}_{discharge} + \text{PV} + \text{Gen} + \text{Grid} + \text{LPS} \quad (4.2.1)$$

where the left and right-hand side of the equation should be approximately the same. Small differences in (4.2.1) can occur as a result of rounding-off errors.

When ϵ is large, i.e. close to 1, the actions are randomised. The RL-based controller thus starts off as a random controller. As the agent learns the environment through exploration, it starts to observe how the state-action space interacts. The RL-based controller improves over the episodes. The comparison between the random and RL-based controller highlights how the RL agent has learned and improved its policies through reinforcement. The RL controller has a lower generator usage, which would result in lower fuel costs. The grid is higher, but the grid's cost of energy is still cheaper than the generator. The PV and battery usage are also higher than the random controller, which indicates that the agent has learned to use the RES and exchange the energy to the battery storage.

The training set and validation set are the basis of how the model is determined. The test set is a completely 'new' dataset that the NN has never seen before. In this way, a more accurate representation of how well the model actually performs can be obtained. The results are shown in Table 4.6. The RL's rewards are higher than the rule-based, whereas the validation it was slightly higher. The rule-based controller outperformed the RL-based and random action controller with an overall reduced LPS and higher RES utilisation. The RL control system was able to effectively exchange the energy between the battery, load and other power sources. The RL-controller was optimised for minimising the LPS, as well as maximising the cost optimisation. The RL-based controller shows promising results and further research can increase the effectiveness of the controller.

Table 4.6: Test Baseline Comparison of Control System: RL

	Rewards	LPS [kWh]	Battery Usage [kWh]	PV Usage [kWh]	Generator Usage [kWh]	Grid Usage [kWh]	Battery Charged [kWh]
Random	109158	253.9	305.8	539.7	190.3	330.5	305.2
Rule	140337	12.1	577.1	1203.5	10.7	82.5	571.1
RL	150130	54.6	524	1001.6	54.5	197.8	517.7

4.2.3 Reinforcement Learning-based and IoT Controller

Seven months (1 March 2019 to 30 September 2019) were used for the RL and IoT-based controller. The database allowed for the largest consecutive hourly scraped data to be used for analysis. The weather may be an influence as the months are usually more overcast as the weather transitions from autumn to winter and then the start of spring. It should be noted that the dataset for the RL-IoT controller is considerably smaller and thus over-fitting may occur. The RL-only controller is also added to see if the additional weather predictions improved the controller.

Table 4.7 shows the validation of the model, which showed that the rewards for the RL and IoT controller was the highest. The RL-only model was also used as part of the results. This is to analyse whether the IoT-information was used by the NN. The RL with IoT control system had a higher reward and lower LPS than the RL-only controller, as well as the highest PV usage. The results showed that the rule-based controller had the lowest LPS. The LPS is decreased by 10.6 kWh (20.7%) and the PV usage is increased by 31 kWh (6.93%) from the RL to the RL-IoT application.

Table 4.8 shows the test results of the two baselines, the RL-based controller and the RL-IoT-based controller. The results between the RL and RL-IoT showed that the IoT-application reduced the LPS by 8.5 kWh (12.7%) and increased the PV usage by 36.6 kWh (7%). This shows that the RL-agent benefited from using the IoT-information.

Table 4.7: Validation Baseline Comparison of Control System: RL and IoT

	Rewards	LPS [kWh]	Battery Usage [kWh]	PV Usage [kWh]	Generator Usage [kWh]	Grid Usage [kWh]	Battery Charged [kWh]
Random	56912	134.5	152.3	256.1	80.4	170.4	152.3
Rule	74042	7.42	294.4	284.1	7.1	40.1	291.8
RL	75965	51.2	249.5	447.1	34.2	106.0	246.6
RL and IoT	76326	40.6	265.8	478.1	32.0	87.9	263.0

Table 4.8: Test Baseline Comparison of Control System: RL and IoT

	Rewards	LPS [kWh]	Battery Usage [kWh]	PV Usage [kWh]	Generator Usage [kWh]	Grid Usage [kWh]	Battery Charged [kWh]
Random	66316	147.7	203.2	329.6	110.8	181.5	203.0
Rule	84778	12.1	364.0	692.5	9.11	52.1	360.0
RL	87823	67.1	302.0	521.7	43.4	133.6	297.8
RL and IoT	87845	58.6	328.5	558.3	34.8	114.2	324.5

4.2.4 Discussion

The control system analysis showed that the RL-based controller is able to minimise the LPS and optimise the cost by using the RES and battery more over the generator and grid. By reinforcement only, with rewards for good behaviour, the controller learns how to exchange the energy in the system. The control system effectively learns policies using the Q-learning method to achieve these goals.

The RL controller does not outperform the rule-based controller. The RL controller had a higher rewards value than the rule-based, which indicates that the generator use is optimised. The results do however indicate that the RL-based controller has learned to minimise the LPS while optimising the cost of the system by learning to reduce the generator usage and increase the RES. This can be seen from how the results improved from the random controller, where the RL initially starts, to the RL's final results. The cyclical nature of the load profile allows the RL agent to study the load profile. The weather gradually changes over the seasons and the agent can also study this change and adapt. The model can be improved by using a larger dataset and simulate different loads to increase its adaptivity. Different intervals can also influence the controller's behaviour. However, consideration has to be considered for practically implementing the RL-based controller. It might be beneficial to consider, for example, minutely actions, but only train every 15 minutes. In doing so, the computational power is not wasted unnecessary.

The rule-based controller does not adapt to sudden changes in the load, whereas the RL controller can learn to adapt. The rule-based controller outperforms the RL-controller in terms of minimising the LPS and reducing the generator use more, and therefore can be seen as a more viable control system. Further studies have to be done in order the improve the performance of RL-based control system. One advantage of RL over rule-based is that a system model is not required. A larger dataset can increase the RL-based controller's performance. This research proves that the concept of RL-based control is worth pursuing.

An optimal controller can reduce the investor's initial capital costs. The effi-

ciency and effectiveness of the controller can increase the utilisation of power sources and this can reduce overall costs. The design of an HPS can be done and then further optimised with a RL controller. The optimised controller can reduce the size of the HPS required to satisfy the load. The design is then reiterated to take into account the controller's efficiency to reduce overall capital costs and increase the ROI of the investor.

Incorporating the weather predictions from the yr.no website increased the PV utilisation. This is an indication that the predicted weather input gave the agent additional information to use to form its policies. This addition increased the use of the RES, which can reduce the operational costs of the generator and grid.

Chapter 5

Conclusions and Recommendations

The combination of power sources, especially when intermittent and stochastic resources such as RESs are included, presents a non-linear optimisation problem in designing an HPS. Different performance indicators start to trade off against one another and a compromise between different aspects have to be determined. A GA was implemented to solve this problem. This design algorithm was utilised as a tool to aid the investor's decision-making process. The different attributes of the HPS configurations are considered and highlighted to indicate how the investor should consider an optimal HPS design. The results of the GA is compared to commercially available software HOMER to assess its validity as a design method.

As the combination of power sources increases, the complexity of the energy exchange in the system also increases. A model-free Q-learning controller is incorporated to increase the system's effectiveness and efficiency. The controller's objectives are to minimise the loss of power supply while optimising the cost of the HPS. To further increase the capabilities of the RL-controller, an IoT-approach is also considered by including the weather predictions from the Norwegian website yr.no.

In chapter 1, the background of HPSs are given to provide the reader insight into advantages and the problems associated with the design and control of an HPS. This analysis gave insight into creating clear research goals and objectives. Chapter 2 gives the relevant theory and relevant work for the reader in how the research methodology aims achieve these objectives set about in Chapter 1. The experimental design is critically discussed in Chapter 3 which gives a practical guide to how the research objectives can be realised. The experimental design produced results which is analysed in Chapter 4. This gave insight into conclusions which can be drawn from the results and to provide recommendations for future work.

5.1 Conclusions

5.1.1 Comparison of Genetic Algorithm and HOMER Software

A GA was used as a possible design solution for an HPS. Using a case study, the results showed that a GA is a feasible method to design an HPS. A trade-off of various attributes was done to highlight the differences between various HPS configurations. Four HPS configurations were analysed and a trade-off analysis of the these attributes was discussed. The attributes include the technical, financial and environmental objectives such as the loss of power supply probability, return on investment, PV and generator utilisation. Projections based on the one-year analysis of the HPS was done to assess the feasibility of the system throughout its lifetime and is taken into account by the algorithm. The algorithm does not only consider a short-term analysis, but also the long term potential of an HPS configuration. The algorithm considers a diverse set of component types to assess the most suitable configuration and component capacity. This is also included in the analysis to give insight into the short-term and long-term feasibility of the system.

In a separate design, the software package HOMER was used as a design tool. HOMER is considered an industry standard for the design of integrated power systems. The GA generates more diverse results with regards to the component types and sizes, whereas HOMER only gives the overall size and one type of power source. The trade-off between the two design methods is that the GA has to be built, which may be time-consuming, compared to the readily available commercial software. However, the GA can be modified to user specifics, such as specifying control constraints and fill in missing data.

It is interesting to note how the addition of a diesel generator with back-up battery storage considerably increases the capacity factor and reduced the LPSP of the system. This is how the HPS can be of vital importance in ensuring future sustainable energy production. Instead of completely eliminating any forms of fossil fuel, incorporating RESs can significantly decrease the dependency on these fossil fuels. The end goal is to generate enough power to satisfy the load, but not at the cost of having no power. The generator is used as a back-up when the storage is depleted and/or when no solar power is available. The exploitation of these advantages of HPSs can have a significant positive impact on areas with no access to power.

Both the GA and HOMER methods showed that more power sources resulted in a lower capacity shortage. This is concurrent with how the weakness of one power source is replaced with another. The SGBG configuration had battery storage to store excess generated PV power, as well as increasing the load factor for the generator to run more fuel-efficient, according to the fuel curve.

The diesel generator could also account for large load peaks during the year. Instead of running a generator the entire year, the main use of power would be the PV modules, battery storage and grid and the generator would offset the peak loads. This leads to a more environmentally friendly HPS, compared to one which uses only generators, by reducing carbon emissions. The GA also considers the LPSP during critical and non-critical hours of the day, which may influence the investor's decision. The configurations of the GA are more diverse, viable and robust and therefore can be seen as a more accurate and faster design approach than HOMER.

5.1.2 Reinforcement Learning-based Control System

A model-free Q-learning reinforcement learning control system for an HPS was done. A case study using a load profile and a weather profile indicated that the RL-based controller did not outperform the rule-based controller, but did outperform the random action controller. The random action controller is the starting point for the RL-based controller in the initial stages of exploration. As the RL agent learns through rewards and penalties, it starts to formulate policies for state-action pairs. The agent attempts to maximise the possible future discounted rewards. This method is known as Q-learning. The Q refers to the 'quality' of the action. An action with a higher Q-value in a specific state will be more likely to be executed in the control system. The agent learns with no prior knowledge of the system and adapts its policies to find an optimal control solution. The RL-based controller was done using Keras in Python3.

Intervals of different control actions were chosen as 15 minutes to increase the training efficiency of the NN. The RL-based controller has achieved its goals of learning the system, minimising the LPS and optimising the cost by reducing expensive sources such as the generator and grid. The controller does not exclude these options as the LPS takes precedence over the cost optimisation, but only after all other alternatives have been explored. The RL-based controller had a higher reward value than the rule-based controller, which indicates that the use of the generator was optimised when the PV was zero. In terms of whether RL-based is better than the rule-based controller, remains inconclusive and further studies have to be done to verify this. The RL-based method has been proven as a feasible control method and the practical implementation would be able to give considerable insight into how the HPS control system can be improved using ML. The RL-controller can adapt autonomously, whereas the rule-based controller has to be manually adapted. The RL-controller can adjust as changes in the load profile and weather occur. Changes such as droughts or rain-heavier winters occur slowly over time and the RL-controller can adapt accordingly. The rule-based controller has a predefined set of rules and thus cannot execute optimal actions in these unknown

and new environments.

Weather predictions were added to the controller to provide additional information with regards to future solar insolation. The weather predictions were taken from the Norwegian weather prediction website, yr.no. The concept was first proven using a simple linear regression to show that the solar irradiance can be linearly calculated from the weather predictions. A database of scraped data had to be created as no such database is available. The results showed a linear relationship between the different input parameters obtained from the yr.no website and the actual solar irradiance.

These weather predictions were then added to an RL-controller in hoping to improve the controller by increasing the use of the RES. The weather predictions database constrained the size of usable data points and only seven months of data were used as part of the analysis. The addition of the weather input showed an slight increase in PV usage, decrease in generator usage and a reduction in the LPS. The RL-IoT controller did not outperform the rule-based controller but has given valuable insight into how the IoT-industry can be exploited to increase the effectiveness of control systems.

5.2 Recommendations

5.2.1 Design of a Hybrid Power Supply

A larger database of components, as well as more types, can be used to refine the GA's results. In doing so, more configuration possibilities can be explored to find an optimal solution. Other system variables, such as the generator's minimum running hours, PV-module tilt angle and the battery's DOD, can be optimised to further enhance the design of the HPS. Tracking PV-modules can also increase the yield which can also reduce the costs of the system and this implementation can be considered as another insight in the same manner as the LPSP critical is considered. The selling of excess generated energy can also be considered, which can influence the financial objectives of the algorithm.

The algorithm is completely reliant on the information it is fed, and thus the load and weather data has to come from a reliable source. Sensitivity analysis may give valuable insight into possible future scenarios, such as carbon tax laws, escalation of fuel prices above inflation and improving system components. Other power configurations can be considered as part of the design. These power sources can include biomass, biogas and wind turbines and alternative energy storage such as hydro- and ultra-capacitor storage.

Alternative design algorithms and software, as discussed in Chapter 2, can be used as a comparison tool to the GA. The control system used in the design process can be improved, which will give a more accurate representation of

how the HPS operates. An optimised control system may reduce the required components, reduce costs and increase the ROI for the investor.

5.2.2 Control of a Hybrid Power Supply

The RL agent's performance is mostly dependent on the reinforcements it receives. Different rewards and penalties can be further explored to reduce the LPS and increase the cost effectiveness of the controller. Greater hardware capabilities and computational power can give more insight into the performance of the controller by storing more information within the simulation. The additional information can be analysed to further improve the RL-controller.

Different network architectures can be implemented and the hyperparameters can be further optimised. Long short-term memory (LSTM), a recurrent neural network (RNN) architecture, can also be considered as a possible control system mechanism. Other RL methods can be investigated. A more refined reward system can significantly improve the quality of the RL-controller. Different optimisation and activation functions can be considered to improve the design of the control system. Hardware constraints and the physical control system is also important. Reinforcement learning, especially in the control system sense, requires more CPU than GPU, which can become expensive. Thus there is always a trade-off between the ideal results and realistic results. A very simple model of the HPS was used in the rule-based control system. An accurate model of the HPS can be developed, which will result in a model-based control system. Other control methods can be investigated as part of a comparison for the RL control system.

The integration of an on-line HPS can be the next phase of RL-based applications. There has to be some form of redundancy to allow the RL controller to make mistakes, but not that it results in a complete LPS during the training on the controller. As much as possible training has to be done before the implementation is done and the controller is gradually given more exercise to control over the HPS. Thus a rule-based system can be initially implemented and as the controller starts to learn on its own, the RL-based controller is given more exercise to control the system. Because of the real-life implications that can occur when an RL-based controller trains, such as a loss of power supply, it is of vital importance that the implementation is done as smoothly as possible. This will mitigate the ethical risks involved when using artificial intelligence as a control system.

When incorporating an IoT-based system, it makes sense to exploit systems which have already done the calculation work and use this information to improve other systems. However, a redundancy has to be ensured to be prepared for power failures or network errors where the controller cannot contact the server. To a certain extent, it should be able to function offline and online,

where the online IoT-based system improves the computational efficiency. This efficiency can be measured in terms of using less power to update the control system after every time interval and to retrain the NN again. It seems unnecessary to have a control system which will make near-perfect decisions, but uses most of the HPS's energy just to train its neural network architecture. There will thus always be a trade-off when achieving perfection.

Loads tend to be cyclical to a certain extent, which makes RL a possible candidate to learn and control the energy exchange of an HPS. The household load changes slowly as the seasons progress and the RL-based controller can slowly learn this as the agent explores the search space. However, loads with non-cyclical aspects, such as electrical vehicles, where the road profile might change significantly, can have an impact on the system. A load profile which is measured with accurate weather predictions can provide more accurate results. Predictions further into the future, such as 1-3 hours, can add additional input for the agent to consider. This may improve the minimisation of the long-term LPS. The RL with IoT controller has a very small database and is not representative of one year of seasonal fluctuations. This can be improved by creating a larger database and simulating the results again. The data for this thesis is not linked in terms of how they are measured. The load profile is simulated with the statistical background of the location's weather profile, but not the actual minute-by-minute measurements.

Appendices

Appendix A

Solar Irradiance Models

A.1 Clear sky model

PVLib is a Python library which can be used to model solar PV-modules [52]. The Location object has a clear sky model. The Location object requires the latitude, longitude, timezone and altitude. The co-ordinates for Stellenbosch is specified as -33.9346° latitude, 18.8668° longitude and the altitude is 122 meters above sea level. The timezone is specified as Johannesburg. The library has a built-in function which can be called to calculate the clear sky model. The returned information is given as a Pandas data frame. Pandas is useful an open-source software library for the manipulation and analysis of data [53]. When the function is called, it returns the clear sky GHI, DNI and DHI values for different times.

A.2 Irradiance calculations

PVLib also has an irradiance class which can calculate the total irradiance. The function returns the plane of array global, direct, diffuse, sky diffuse and ground diffuse irradiance. Irradiance refers to the instantaneous measure of power and is measured in W/m^2, whereas insolation is solar energy in kWh.

The function requires the surface tilt, surface azimuth-, solar zenith- and solar azimuth angle, as well as the albedo, model, DNI, GHI and DHI. These parameters will be discussed below. The isotropic model is used in these calculations.

Surface tilt angle The surface tilt Σ refers to the collector's angle which it is tilted up. The tilt angle is fixed at 30°.

Surface azimuth angle The surface azimuth is the angle measured from due north in the southern hemisphere. The collector is North-facing and thus

the surface azimuth angle ϕ_C is 0°.

Solar zenith angle The solar zenith angle θ_S is the complement of the solar altitude angle β and is calculated by (A.2.1):

$$\cos\theta_S = \sin L \sin\delta + \cos L \cos\delta \cos H \tag{A.2.1}$$

The function requires the latitude, hour angle and declination angle. The hour angle H is calculated using (A.2.2) and the declination angle δ is calculated using (A.2.5).

$$\text{Hour angle } H = \left(\frac{15^\circ}{h}\right) \cdot (\text{Hours before solar noon}) \tag{A.2.2}$$

The hours before solar noon is equated using the Equation of Time, expressed by (A.2.3).

$$E = 9.87 \sin 2B - 7.53 \cos B - 1.5 \sin B \tag{A.2.3}$$

$$B = \frac{360}{364}(n_{day} - 81) \tag{A.2.4}$$

where n_{day} refers to the number of day of the year. The solar declination angle is expressed as:

$$\delta = 23.45 \left(\frac{2\pi}{365}(n_{day} + 284)\right) \tag{A.2.5}$$

Solar azimuth angle Solar azimuth angle (ϕ_S) is expressed by (A.2.6).

$$\sin\phi_S = \frac{\cos\delta \sin H}{\sin\theta_S} \tag{A.2.6}$$

and if $\cos H \geq \frac{\tan\delta}{\tan L}$ then $|\theta_S| \leq 90^\circ$, otherwise $|\theta_S| > 90^\circ$.

Incidence angle The incidence angle θ for a fixed orientation is given by (A.2.7).

$$\cos\theta = \sin\theta_S \cos(\phi_S - \phi_C) \sin\Sigma + \cos\theta_S \cos\Sigma \tag{A.2.7}$$

Albedo The ground's albedo ρ refers to the reflectivity of the ground and is an estimated value. Snow, for example is 0.8 whereas gravel is 0.1 and grass is ordinarily 0.2.

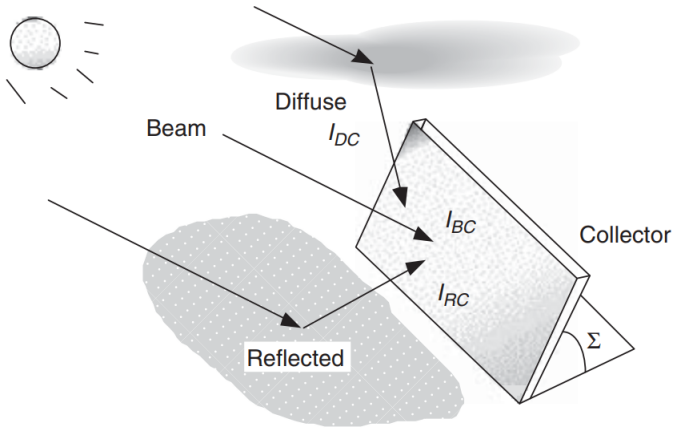

Figure A.1: Solar irradiance components [45]

A.2.1 Irradiance calculations

Figure A.1 shows the different irradiance components, namely direct beam (I_{BC}), diffuse (I_{DC}) and reflected (I_{RC}) irradiance, and the collector tilt angle Σ. These irradiance components are expressed by equations (A.2.8), (A.2.9) and (A.2.10).

$$I_{BC} = \text{DNI} \cos\theta \tag{A.2.8}$$

$$I_{DC} = \text{DHI}\left(\frac{1+\cos\Sigma}{2}\right) \tag{A.2.9}$$

$$I_{RC} = \text{GHI} \cdot \rho\left(\frac{1-\cos\Sigma}{2}\right) \tag{A.2.10}$$

The total irradiance is expressed by:

$$I_C = I_{BC} + I_{DC} + I_{RC}. \tag{A.2.11}$$

The insolation can be calculated from the total irradiance using

$$\bar{I_C} = I_C \cdot \text{Time (h)}. \tag{A.2.12}$$

Appendix B

Data Input

B.1 Components Database

The components for the genetic algorithm consists of PV modules, inverters, generators and batteries. These databases are summarised in Tables B.1, B.2, B.3 and B.4.

Table B.1: Design Database of Inverters

Name	Price (ZAR)	Max Power (kW)	Input Voltage (V) min	Input Voltage (V) max	Warranty (years)	Reference
SMA Sunny Tripower TL15000	55,053	15	240	800	3	[54]
SMA Sunny Tripower TL20000	59,240	20	320	800	3	[55]
SMA Sunny Tripower TL25000	61,166	25	320	800	3	[56]

Table B.2: Design Database of Batteries

Name	Price (ZAR)	Capacity (Ah)	Rated Voltage (V)	60 DOD Life cycles	Reference
OmniPower	3,747	120	12	1800	[57]
OmniPower	5,603	180	12	1800	[58]
OmniPower	6,990	240	12	1800	[59]

Table B.3: Design Database of Generators

Name	Prime Power (kVA)	Rated Voltage (V)	PF	Fuel usage 75% load (L/hour)	Price (ZAR)	Reference
MAC AFRIC	50.0	380	0.8	10.9	129,950	[60]
MAC AFRIC	34.0	380	0.8	9.0	142,500	[61]
MAC AFRIC	37.5	380	0.8	8.5	105,950	[62]

Table B.4: Design Database of PV Modules

Name of Company	Price (ZAR)	Warranty (years)	Panel Area (m^2)	Max Power (W)	MPP Voltage (V)	MPP Current (A)	Reference
Renewsys	4,960	10	1.97	380	39.40	9.70	[63]
Galactic	4,344	10	1.97	365	38.97	9.43	[64]
JA Solar	4,611	10	1.94	330	37.65	8.70	[65]
Canadian	2,327	10	1.92	330	37.20	8.88	[66]
Solar	2,908	10	2.21	405	38.9	10.42	[67]
	2,846	10	2.21	400	38.7	10.34	[68]
	2,971	10	2.21	410	38.7	10.34	[69]
	2,469	10	1.98	360	39.6	9.10	[70]
	2,640	10	2.21	395	38.50	10.26	[71]

B.2 Yr.No Data

The data is received as an ordered dictionary, which has to be pre-processed to use as a usable Pandas data frame. The most important data for the controller is kept and is shown in Table 3.7 in Chapter 3.

A database was accumulated over 13 months to ping the yr.no website every hour, or as much as possible. An analysis of the accuracy of the predictions has to be done to assess the feasibility of using this data. The Pearson correlation coefficient is a measure of how much values are linearly related and is expressed by (B.2.1) and estimated using (B.2.2). If it is closer to 1, it means the correlation is very close, whereas close to 0 means it is uncorrelated. If it is close to -1, it means that the values are negatively correlated, meaning that if the one measurement goes up, the other is expected to decrease.

$$\rho = \frac{\text{cov}(X, Y)}{\sigma_x \sigma_y} \tag{B.2.1}$$

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 (y_i - \overline{y})^2}} \tag{B.2.2}$$

Using this metric, the correlation between the predicted and actual values are assessed in Table B.5. In the table, the following abbreviations are used: CS refers to clear sky, NS refers to normal or actual sky and Pred to predicted. It showed that the predicted and actual measurements are correlated, as well as giving insight into relationships between the different variables.

From Table B.5, a moderately negative correlation can be noted between the pressure and temperature, the humidity and temperature, and humidity and wind speed. Table B.6 shows the correlation matrix between the predicted values received from yr.no. As expected, there is a negative correlation between the temperature and humidity, but a neutral correlation between the clouds and temperature. The wind speed and low cloud percentage are slightly correlated, as well as the humidity and low clouds. The temperature and low clouds are moderately negatively correlated. The low and high clouds are neutrally correlated, but the medium and low clouds are moderately correlated.

In Table B.7, the actual values obtained from SAURAN [11] is analysed. The DNI, DHI and GHI show a moderate correlation with the temperature, indicating that an increase in irradiance will most likely increase the temperature. Since the global irradiance is dependent on the DNI, DHI and GHI, it can be

Table B.5: Correlation Matrix of Actual versus Predicted measurements

	Temperature (T)		Pressure (P)		Humidity (H)		Wind speed (W)	
	NS	Pred	NS	Pred	NS	Pred	NS	Pred
NS (T)	1	0.801	-0.446	-0.484	-0.734	-0.477	0.363	0.263
Pred (T)	0.801	1	-0.436	-0.510	-0.584	-0.742	0.134	0.159
NS (P)	-0.446	-0.436	1	0.973	0.110	0.114	-0.164	-0.209
Pred (P)	-0.484	-0.510	0.973	1	0.160	0.205	-0.132	-0.197
NS (H)	-0.734	-0.584	0.110	0.160	1	0.691	-0.377	-0.258
Pred (H)	-0.477	-0.742	0.114	0.205	0.691	1	-0.067	-0.114
NS (W)	0.363	0.134	-0.164	-0.132	-0.377	-0.067	1	0.695
Pred (W)	0.263	0.159	-0.209	-0.197	-0.258	-0.114	0.695	1

Table B.6: Correlation Matrix of Predicted Values

	Clouds	High clouds	Humidity	Low clouds	Medium clouds	Temperature	Wind speed
Clouds	1.000	0.645	0.296	0.667	0.486	-0.176	0.149
High Clouds	0.645	1.000	-0.047	-0.026	0.123	0.052	-0.007
Humidity	0.296	-0.047	1.000	0.460	0.263	-0.742	-0.114
Low Clouds	0.667	-0.026	0.460	1.000	0.481	-0.287	0.210
Medium Clouds	0.486	0.123	0.263	0.481	1.000	-0.175	0.258
Temperature	-0.176	0.052	-0.742	-0.287	-0.175	1.000	0.159
Wind speed	0.149	-0.007	-0.114	0.210	0.258	0.159	1.000

Table B.7: Correlation Matrix of Actual Values

	DNI	DHI	GHI	Temp-erature	Humid-ity	Wind speed	I_C	I_{BC}	I_{DC}	I_{RC}
DNI	1.000	0.341	0.887	0.600	-0.650	0.249	0.906	0.370	0.341	0.887
DHI	0.341	1.000	0.627	0.345	-0.339	0.290	0.581	0.999	1.000	0.627
GHI	0.887	0.627	1.000	0.625	-0.628	0.316	0.973	0.653	0.627	1.000
Temperature	0.600	0.345	0.625	1.000	-0.717	0.385	0.575	0.363	0.345	0.625
Humidity	-0.650	-0.339	-0.628	-0.717	1.000	-0.389	-0.629	-0.357	-0.339	-0.628
Wind speed	0.249	0.290	0.316	0.385	-0.389	1.000	0.258	0.295	0.290	0.316
I_C	0.906	0.581	0.973	0.575	-0.629	0.258	1.000	0.607	0.581	0.973
I_{BC}	0.370	0.999	0.653	0.363	-0.357	0.295	0.607	1.000	0.999	0.653
I_{DC}	0.341	1.000	0.627	0.345	-0.339	0.290	0.581	0.999	1.000	0.627
I_{RC}	0.887	0.627	1.000	0.625	-0.628	0.316	0.973	0.653	0.627	1.000

expected that an increase in temperature is a result of a sunnier day. The humidity is negatively correlated with the global irradiance, as expected.

Table B.8 shows the correlation matrix between the predicted information, the clear sky irradiance and actual irradiance. The clear sky and normal sky values are strongly correlated, which can be helpful for the NN to predict the actual global irradiance. As expected, the temperature and the irradiance of both clear sky and actual are correlated. The wind speed and the low and medium clouds are moderately correlated, but the high clouds and wind speed are neutrally correlated. The wind speed and irradiance are slightly correlated as well. The actual irradiance is slightly negatively correlated to the low and medium clouds.

Table B.8: Correlation Matrix of Actual and Predicted values with solar irradiance

	Temp-erature	Wind speed	Clouds	Low clouds	Medium clouds	High clouds	CS I_C	NS I_C
Temperature	1	0.159	-0.176	-0.287	-0.175	0.052	0.600	0.650
Winds	0.159	1	0.149	0.210	0.258	-0.007	0.325	0.240
Clouds	-0.176	0.149	1	0.667	0.486	0.645	-0.025	-0.185
Low Clds	-0.287	0.210	0.667	1	0.481	-0.026	-0.054	-0.214
Medium Clds	-0.175	0.258	0.486	0.481	1	0.123	0.013	-0.145
High Clds	0.052	-0.007	0.645	-0.026	0.123	1	0.018	-0.030
CS I_C	0.600	0.325	-0.025	-0.054	0.013	0.018	1	0.898
NS I_C	0.650	0.240	-0.185	-0.214	-0.145	-0.030	0.898	1

Appendix C

Code

C.1 Genetic Algorithm

The implementation of the GA was done using object-oriented programming (OOP). OOP is a programming style and is associated with the concepts of classes, objects, inheritance, encapsulation, abstraction and polymorphism [72]. The programming style reads intuitively, which makes it easier to understand and debug the implementation.

A class is defined with attributes. When an object is defined, an instance of the class is created. The object has specified attributes. For an HPS chromosome class, the types and numbers of the different components vary, but they are still present as part of the attributes of the HPS chromosome.

Basic OOP principles are encapsulation, composition and inheritance. Encapsulation is the concept of data which is inside the object can only be accessed through a public interface, which will be the object's methods. It is defined once in a logical place and not multiple times, which condenses the programming and makes it easier to understand. Composition is a method of aggregating objects together by making some object attribute other objects. For the HPS chromosome class, it has other objects such as the PV-Module, Generator, Inverter, Battery and Grid classes which has their own attributes and functions. The relationships between these compositions can be one-to-one, one-to-many or many-to-many and can also be unidirectional or bidirectional. The HPS will have only one type of PV module, however, the PV module can be found in different HPS configurations. Inheritance arranges objects in a hierarchy form from the most general to most specific objects. A subtype of an object is an object which inherits from another object. A subclass is also known as a child class and a superclass is known as a parent class. Figure C.1 visualises a summary of the OOP used to implement the GA. Each object has a set of attributes which have different values.

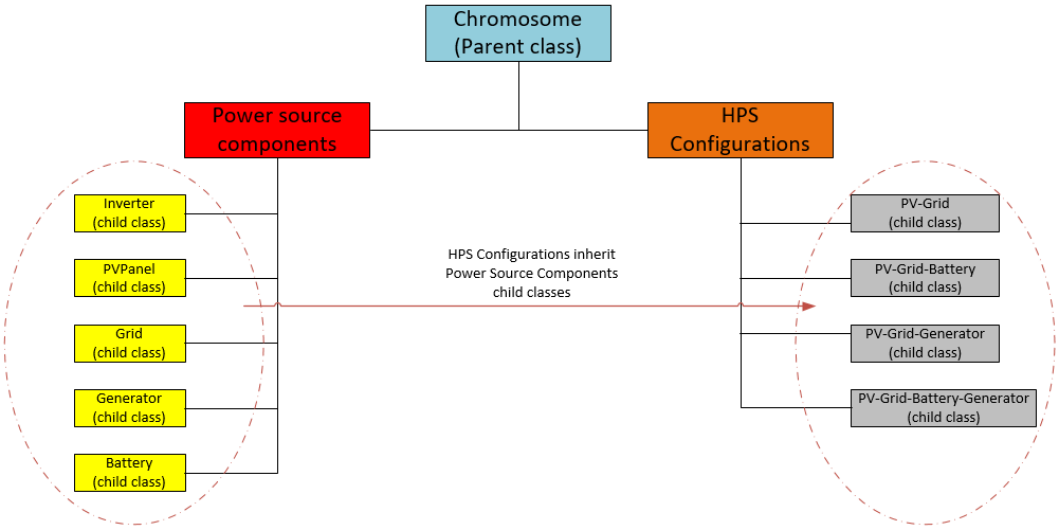

Figure C.1: Genetic Algorithm Object Oriented Programming Summary

The chromosome is the superclass/parent class and has all the information about the chromosome's DNA. The subclasses fall essentially under two categories: the HPS components and the HPS configurations. As seen in Figure C.1, the HPS components are attributed to five classes: PV module, generator, battery, inverter and grid. The HPS configurations, as discussed in section 3.1.4.3, fall under this category. Each of these configurations is defined by their own classes with attributed functions. The HPS configuration classes inherit the component object's attributes. The battery and generators are broken up into smaller arrays of objects to analyse the individual power sources.

C.1.1 Imports

```
In [1]: import warnings
        warnings.filterwarnings('ignore')

        import pandas as pd
        import random
        import csv
        from datetime import datetime,timedelta
        import matplotlib.pyplot as plt
        import math
        from IPython.display import clear_output
        random.seed(30)
        import os

        import numpy as np

        import pvlib
        from pvlib.location import Location, solarposition
```

C.1.2 Graph plot variables

Plot graphs for thesis

```
In [2]: label = 20
        ticklabel = 18

        params = {'axes.labelsize': label,
                'axes.titlesize': label,
                'text.usetex'   : True,
                'font.size': ticklabel,
                'legend.frameon': True,
                'legend.fontsize': ticklabel,
        #         'legend.loc': 'upper right',
                'legend.framealpha': 1,
                'xtick.labelsize': ticklabel,
                'ytick.labelsize': ticklabel,
                'grid.linestyle' : ":",
                'axes.grid': True,
                'axes.grid.which': 'major',
                'axes.grid'      : True,
                'grid.linewidth' : 0.8,
                'keymap.grid': ['g'],
                'keymap.grid_minor': ['G'],
                'figure.figsize': [18, 8]
                }
        plt.rcParams.update(params)
```

C.1.3 Data input preprocessing and analysis

```
In [3]: Weather=pd.read_csv("database/HourlyWeatherData.csv")
        Load=pd.read_csv("database/Plaas.csv")

        Load=Load.drop(['Unnamed: 5','S -->','Q -->'], axis=1)
        Load=Load.rename(index=str, columns={'P -->':'Load'})

        Load['Date']=Load['DATE'].map(str) + ' ' + Load['TIME']
        Load['Date']=pd.to_datetime(Load['Date'])
        Load=Load.drop(['DATE','TIME'],axis=1)

        Load=Load[['Date','Load']]
        Load.head()
```

```
Out[3]:                  Date   Load
        0 2011-01-01 00:00:00  7.251
        1 2011-01-01 00:30:00  6.954
        2 2011-01-01 01:00:00  7.207
        3 2011-01-01 01:30:00  7.013
        4 2011-01-01 02:00:00  6.566
```

```
In [4]: Load.describe()
```

```
Out[4]:                Load
        count  17520.000000
        mean      20.504866
        std       12.624053
        min        0.000000
        25%       10.549000
        50%       16.788500
        75%       29.033250
        max       67.659000
```

```
In [5]: Load['Year']    = Load['Date'].dt.year.copy()
        Load['Month']   = Load['Date'].dt.month.copy()
        Load['Day']     = Load['Date'].dt.day.copy()
        Load['Hour']    = Load['Date'].dt.hour.copy()

        df              = Load.groupby(
            [Load['Year'], Load['Month'],
             Load['Day'], Load['Hour']],
            as_index=False).agg('sum')

        df              = Load.groupby(
            [Load['Year'], Load['Month'],
             Load['Day'], Load['Hour']],
            as_index=False).agg('sum')

        df['Date']      = df['Year'].copy().map(str) \
                        + '-' + df['Month'].copy().map(str) \
                        + '-' + df['Day'].copy().map(str) \
                        + ' ' + df['Hour'].copy().map(str) + ':00'
        df['Date']      = pd.to_datetime(
            df['Date'], format='%Y-%m-%d %H:%M')

        df.head()
```

```
Out[5]:    Year  Month  Day  Hour    Load                Date
        0  2011      1    1     0  14.205 2011-01-01 00:00:00
        1  2011      1    1     1  14.220 2011-01-01 01:00:00
        2  2011      1    1     2  13.144 2011-01-01 02:00:00
        3  2011      1    1     3  13.208 2011-01-01 03:00:00
        4  2011      1    1     4  13.084 2011-01-01 04:00:00
```

```
In [11]: Weather = Weather.rename(index=str,
                                  columns={'TrackerWM_Avg':'DNI',
                                           'SunWM_Avg':'GHI',
                                           'ShadowWM_Avg':'DHI'})

          Weather = Weather.drop(['Date', 'Time',
                                  'Record', 'BattV_Min',
                                  'Tracker2WM_Avg',
                                  'ShadowbandWM_Avg',
                                  'DNICalc_Avg', 'AirTC_Avg',
                                  'AirTC_Max', 'AirTC_Min',
```

```
                                'RH', 'WS_ms_S_WVT',
                                'WindDir_D1_WVT',
                                'WindDir_SD1_WVT',
                                'BP_mB_Avg', 'UVA_Avg',
                                'UVB_Avg'], axis=1)

        Weather['Time'] = pd.to_datetime(Weather['TimeStamp'])

        Weather = Weather.drop(['TimeStamp'], axis=1)
```

C.1.4 Genetic Algorithm Inputs

```
In [16]: population       = 60
         generations      = 80
         crossoverProb    = 0.8
         DNA_MutationProb = 0.4
         years            = 20

         tiltAngle=30
         rho=0.2
         LTM=-30
         latitude=-33.9346
         longitude=18.8668
         altitude=122

         offspringCount = 0

         loc=Location(latitude,
                      longitude,
                      'Africa/Johannesburg',
                      altitude,
                      'Stellenbosch')
```

C.1.5 Search space limits

```
In [17]: minInv, maxInv               = 10, 40
         minBatStrings, maxBatStrings = 10, 20
         minGen, maxGen               = 1, 3

         DOD             = 0.6
         SystemVolt      = 120
         minRunningHours = 2
         derateFactor    = 0.7
```

C.1.6 Financial Analysis Inputs

```
In [18]: inflation = 0.05

         # Operational and maintenance
         omPV  = 0.05
         omBat = 0.06
```

```
omGen = 0.10

installationCost = 0.2
fuelPrice        = 16
gridPrice    = 1.45 # per kWh
```

C.1.7 Mathematical Modeling

C.1.7.1 PV Power

```
In [19]: class PVPanel:
             def __init__ (self, name, Price, warranty, PanelWidth,
                           PanelLength, Power, MaxVolt, MaxCurrent,
                           OCVolt, SCCurrent, Eff, NOCT):
                 self.Name        = name
                 self.Price       = Price
                 self.warranty    = warranty
                 self.PanelWidth  = PanelWidth
                 self.PanelLength = PanelLength
                 self.Power       = Power
                 self.MaxVolt     = MaxVolt
                 self.Eff   = Eff
                 self.area        = self.PanelWidth*self.PanelLength
```

```
In [20]: PVDF = pd.read_csv('database/PVPanelData.csv')
         PVDF = PVDF.as_matrix()
         PVDB = []

         for i in range (0, len(PVDF)):
             PVDB.append(PVPanel(PVDF[i][0],
                                 PVDF[i][1],
                                 PVDF[i][2],
                                 PVDF[i][3],
                                 PVDF[i][4],
                                 PVDF[i][5],
                                 PVDF[i][6],
                                 PVDF[i][7],
                                 PVDF[i][8],
                                 PVDF[i][9],
                                 PVDF[i][10],
                                 PVDF[i][11]))
```

```
In [21]: WeatherFile='database/HourlyWeatherData.csv'

         WeatherDF=pd.read_csv(WeatherFile)

         WeatherDF=WeatherDF.drop(['Record', 'Date', 'Time',
                                   'UVA_Avg', 'UVB_Avg',
                                   'Tracker2WM_Avg',
                                   'WindDir_D1_WVT',
                                   'WindDir_SD1_WVT',
                                   'ShadowbandWM_Avg',
```

```
                                    'AirTC_Max', 'DNICalc_Avg',
                                    'BattV_Min', 'WS_ms_S_WVT',
                                    'RH', 'BP_mB_Avg',
                                    'AirTC_Avg', 'AirTC_Min'],
                                   axis=1)

        WeatherDF=WeatherDF.rename(index=str,
                                   columns={'SunWM_Avg':'GHI',
                                            'TrackerWM_Avg':'DNI',
                                            'ShadowWM_Avg':'DHI',
                                            'TimeStamp':'Date'})

        WeatherDF['Date']=pd.to_datetime(WeatherDF['Date'])
```

```
In [22]: def getTotalIrradiance(GHI, DNI, DHI, Time):
             Time = pd.DatetimeIndex(Time)
             Time = Time.tz_localize(loc.tz)
             """
             Calculate the total irradiance

             Args:
                 GHI: global horizontal irradiance
                 DNI: direct normal irradiance
                 DHI: diffuse horizontal irradiance
                 Time: DatetimeIndex

             Returns:
                 total: [poa_global,
                 poa_direct,
                 poa_diffuse,
                 poa_sky_diffuse,
                 poa_ground_diffuse]
             """

             model='isotropic'

             # Panel azimuth from North
             surface_azimuth = 0

             albedo = rho #surface albedo

             # convert to radians
             latitudeR = np.deg2rad(latitude)
             n = Time.dayofyear
             # Equation of time
             """
             pvlib.solarposition.equation_of_time_pvcdrom
                 Parameters: dayofyear
                 Return: equation_of_time: in minutes
             """
             eqOfTime=pvlib.solarposition.equation_of_time_pvcdrom(n)
```

```
# Hour angle
"""
pvlib.solarposition.hour_angle
    Parameters: times: pd.DatetimeIndex must be
    localized to timezone for the longitude
                longitude: longitude (degrees)
                equation_of_time: equation of time in minutes
    Returns: hour_angle (degrees)
"""
hourangle=pvlib.solarposition.hour_angle(Time,
                                         longitude,
                                         eqOfTime)
hourAngleR=np.deg2rad(hourangle)

#Declination angle calculations
"""
pvlib.solarposition.declination_cooper69
    Parameters: dayofyear
    Returns: declination angle (radians)
"""
declinationR = pvlib.solarposition.declination_cooper69(n)

#Solar zenith angle calculations
"""
pvlib.solarposition.solar_zenith_analytical
    Parameters: latitude: Latitude of location (radians)
                hourangle: Hour angle in local solar time (rad)
                declination: declination of the sun (radians)
    Returns: solar zenith angle (radians)
"""
solar_zenithR = pvlib.solarposition.solar_zenith_analytical(
    latitudeR, hourAngleR, declinationR)

#Solar azimuth angle calculations
"""
pvlib.solarposition.solar_azimuth_analytical
    Parameters: latitude: Latitude of location (radians)
                hourangle: Hour angle in local solar time (rad)
                declination: declination of the sun (radians)
                zenith: solar zenith angle (radians)
    Returns: Solar azimuth angle (radians)

 """
solar_azimuthR=pvlib.solarposition.solar_azimuth_analytical(
    latitudeR, hourAngleR, declinationR, solar_zenithR)
#convert to degrees
solar_azimuth = np.rad2deg(solar_azimuthR)
solar_zenith  = np.rad2deg(solar_zenithR)
surface_tilt  = tiltAngle

# get total irradiance
"""
```

```
            pvlib.irradiance.get_total_irradiance
                Parameters: surface_tilt: Panel tilt from horizontal (deg)
                            surface_azimuth: Panel azimuth from north (deg)
                            solar_zenith: Solar zenith angle (degrees)
                            dni: Direct Normal Irradiance (W/m^2)
                            ghi:Global horizontal irradiance (W/m^2)
                            dhi: Diffuse horizontal irradiance (W/m^2)
                            dni_extra: Extraterrestrial DNI.
                            airmass: Airmass
                            albedo: Surface albedo 0-1
                            surface_type: Surface type.
                            model: isotropic(default)
                Returns: dataframe array:
                            poa_global,
                            poa_direct,
                            poa_diffuse,
                            poa_sky_diffuse,
                            poa_ground_diffuse
            """
            total = pvlib.irradiance.get_total_irradiance(
                surface_tilt=tiltAngle, surface_azimuth=surface_azimuth,
                solar_zenith=solar_zenith, solar_azimuth=solar_azimuth,
                dni=DNI, ghi=GHI, dhi=DHI, albedo=albedo, model=model)

            return total


In [23]: total=getTotalIrradiance(
            WeatherDF['GHI'], WeatherDF['DNI'],
            WeatherDF['DHI'], pd.DatetimeIndex(WeatherDF['Date']))

         WeatherDF['Global Insolation']=total['poa_global']

         WeatherDF=WeatherDF.drop(['DNI', 'DHI', 'GHI'], axis=1)

         WeatherDF=WeatherDF.as_matrix()


In [24]: class Weather:

            def __init__(self, Time, insolation):
                self.Time = Time
                self.insolation = insolation

            def __str__(self):
                return ('{} : {}'.format (self.Time, self.insolation))

         WeatherDB = []

         for i in range (0, len(WeatherDF)):
            WeatherDB.append(Weather(
                WeatherDF[i][0], WeatherDF[i][1]))
```

C.1.7.2 Grid Input

```
In [25]: class Grid:

             def __init__(self, kWh, reliable):
                 self.kWh=kWh
                 self.reliable=reliable

         maxGrid = 10 # kWh
         reliable=0.99

         grid=Grid(maxGrid, reliable)
```

C.1.7.3 Battery Model

```
In [26]: class Bat:

             def __init__(self, Name, Price, Ah, ratedV):
                 self.Name=Name
                 self.Price=Price
                 self.Ah=Ah
                 self.ratedV=ratedV
                 self.designLife=10 # years
                 self.Power=Ah*ratedV/1000

         BatDF = pd.read_csv('database/BatteryData.csv')
         BatDF = BatDF.as_matrix()
         BatDB = []

         for i in range (0, len(BatDF)):
             BatDB.append(Bat(
                 BatDF[i][0], BatDF[i][1],
                 BatDF[i][2], BatDF[i][3]))
```

C.1.7.4 Generator Model

```
In [27]: class Gen:

             def __init__ (self, Name, kVA, Voltage, PF,
                           Phase, RPM, tankSize, fuel75Load, Price):
                 self.Name=Name
                 self.kVA=kVA
                 self.Voltage=Voltage
                 self.RPM=RPM
                 self.PF=PF
                 self.Phase=Phase
                 self.Price=Price
                 self.tankSize=tankSize
                 self.fuel75Load=fuel75Load
                 self.Power=self.kVA*self.PF*0.75
                 self.runningTime=self.tankSize/self.fuel75Load
```

```
        def __str__(self):
            msg='{2}: {0:.2f} kW\tRunningTime: {1:.2f} hours'
            msg=msg.format(self.Power, self.runningTime, self.Name)
            return (msg)


GenDF = pd.read_csv('database/GeneratorData.csv')
GenDF = GenDF.as_matrix()
GenDB = []

for i in range (0, len(GenDF)):
    GenDB.append(Gen(GenDF[i][0], GenDF[i][1],
                     GenDF[i][2], GenDF[i][3],
                     GenDF[i][4], GenDF[i][5],
                     GenDF[i][6], GenDF[i][7],
                     GenDF[i][8]))
```

C.1.7.5 Load Profile

```
In [28]: class Load:

            def __init__(self, Time, loadkW):
                self.Time=Time
                self.loadkW=loadkW

            def __str__(self):
                msg='{0}\t{1:.3f} kW'
                msg=msg.format(self.Time, self.loadkW)
                return (msg)

        LoadFile='database/Plaas.csv'

        Load_df=pd.read_csv("database/Plaas.csv")

        Load_df=Load_df.drop(['Unnamed: 5','S -->','Q -->'], axis=1)
        Load_df=Load_df.rename(index=str, columns={'P -->':'Load'})

        Load_df['Date']=Load_df['DATE'].map(str) + ' ' + Load_df['TIME']
        Load_df['Date']=pd.to_datetime(Load_df['Date'])
        Load_df=Load_df.drop(['DATE','TIME'],axis=1)

        Load_df['Year']  = Load_df['Date'].dt.year.copy()
        Load_df['Month'] = Load_df['Date'].dt.month.copy()
        Load_df['Day']   = Load_df['Date'].dt.day.copy()
        Load_df['Hour']  = Load_df['Date'].dt.hour.copy()

        df = Load_df.groupby([Load_df['Year'],
                              Load_df['Month'],
                              Load_df['Day'],
                              Load_df['Hour']],
                             as_index=False).agg('sum')
```

```
df = Load_df.groupby([Load_df['Year'],
                      Load_df['Month'],
                      Load_df['Day'],
                      Load_df['Hour']],
                     as_index=False).agg('sum')

df['Date'] = df['Year'].copy().map(str) \
           + '-' + df['Month'].copy().map(str) \
           + '-' + df['Day'].copy().map(str) \
           + ' ' + df['Hour'].copy().map(str) + ':00'

df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d %H:%M')

df=df.drop(['Year', 'Month', 'Day', 'Hour'], axis=1)

Load_=df.as_matrix()
LoadDB = []

for i in range (0, len(Load_)):
    LoadDB.append(Load(Load_[i][1], Load_[i][0]))

criticalHoursDB=[False, #00:00-00:59
                 False, #01:00-01:59
                 False, #02:00-02:59
                 False, #03:00-03:59
                 False, #04:00-04:59
                 True, #05:00-05:59
                 True, #06:00-06:59
                 True, #07:00-07:59
                 True, #08:00-08:59
                 True, #09:00-09:59
                 True, #10:00-10:59
                 True, #11:00-11:59
                 True, #12:00-12:59
                 True, #13:00-13:59
                 True, #14:00-14:59
                 True, #15:00-15:59
                 True, #16:00-16:59
                 True, #17:00-17:59
                 True, #18:00-18:59
                 True, #19:00-19:59
                 True, #20:00-20:59
                 True, #21:00-21:59
                 True, #22:00-22:59
                 False] #23:00-23:59
```

C.1.7.6 Inverter

```
In [29]: class Inv:

             def __init__(self, Name, Price, Power, MPPT_min,
                          MPPT_max, Eff, Warranty):
```

```
            self.Name = Name
            self.Power = Power
            self.Price = Price
            self.MPPT_min = MPPT_min
            self.MPPT_max = MPPT_max
            self.Eff = Eff
            self.warranty = Warranty

        def __str__(self):
            priceFormat='R{:,.2f}'.format(self.Price)
            msg='{0}\t{1} kW\t{2}\t{3}\t{4}\t{5}%'
            msg=msg.format(self.Name, self.Power, priceFormat,
                           self.MPPT_min, self.MPPT_max,
                           self.Eff)
            return (msg)

    InvDF = pd.read_csv('database/InverterData.csv')
    InvDF = InvDF.as_matrix()
    InvDB = []

    for i in range (0, len(InvDF)):
        InvDB.append(Inv(
            InvDF[i][0], InvDF[i][1],
            InvDF[i][2], InvDF[i][3],
            InvDF[i][4], InvDF[i][5],
            InvDF[i][6]))
```

C.1.8 HPS Configurations

C.1.8.1 Chromosome

```
In [30]: class Chromosome:

            def setTypes(self, PV, bat, gen, inv):
                self.typePV  = returnType(PV)
                self.typeBat = returnType(bat)
                self.typeGen = returnType(gen)
                self.typeInv = returnType(inv)

            def setSizes(self, SystemVolt, minBatStrings,
                         maxBatStrings, minGen, maxGen,
                         minInv, maxInv):
                self.numInv = returnNumber(minInv, maxInv)
                self.setPVSize()
                self.numGen        = returnNumber(
                    minGen, maxGen)
                self.numBatStrings = returnNumber(
                    minBatStrings, maxBatStrings)
                self.setBatSize(SystemVolt)

            def setBatSize(self, SystemVolt):
                self.seriesBat     = int(math.floor(
```

```
            SystemVolt/self.Bat.ratedV))
        self.numTotalBat   = self.numBatStrings*self.seriesBat
        self.systemVoltage = self.Bat.ratedV*self.seriesBat

    def setPVSize(self):
        minPVString = math.ceil(
            self.Inv.MPPT_min/self.PVPanel.MaxVolt)
        maxPVString = math.ceil(
            self.Inv.MPPT_max/self.PVPanel.MaxVolt)
        self.numPV  = (
            returnNumber(minPVString,
                         maxPVString))*self.numInv

    def setOffspring(self, TPV, TBat, TGen, TInv, NBatS,
                     NGen, NInv):
        self.typePV        = TPV
        self.typeBat       = TBat
        self.typeGen       = TGen
        self.typeInv       = TInv
        self.numBatStrings = NBatS
        self.numGen        = NGen
        self.numInv        = NInv

    def setClasses(self, PVPanel, Bat, Gen,
                   Inv, Grid):
        self.PVPanel    = PVPanel
        self.Bat    = Bat

        self.Gen = Gen
        self.Inv = Inv
        self.Grid       = Grid

    def setPower(self):
        self.PVCapacity  = self.numPV*self.PVPanel.MaxPower
        self.batCapacity = self.numTotalBat*self.Bat.Power
        self.genCapacity = self.numGen*self.Gen.kVA

    def setFitness(self):
        setFitness(self.SG)
        setFitness(self.SGB)
        setFitness(self.SGG)
        setFitness(self.SGBG)

        self.fitness = self.SG.fitness \
                     + self.SGB.fitness \
                     + self.SGG.fitness \
                     + self.SGBG.fitness

    def rouletteWheel(self, start, end):
        self.startWheel = start
        self.endWheel   = end
```

```
            def EconomicalAnalysis(self):
                economicalAnalysis(self.SG)
                economicalAnalysis(self.SGB)
                economicalAnalysis(self.SGG)
                economicalAnalysis(self.SGBG)

            def printChromosomeInfo(self):
                msg='{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t'.format(
                    self.typePV, self.typeBat, self.typeGen,
                    self.typeInv, self.numBatStrings, self.numGen,
                    self.numInv)
                return (msg)

            def setHPS(self):
                self.SG = HPS_PVGrid(
                    self.PVPanel, self.numPV, self.Grid,
                    self.Inv, self.numInv)

                self.SGB = HPS_PVGridBat(
                    self.PVPanel, self.numPV, self.Grid,
                    self.Inv, self.numInv, self.Bat,
                    self.numBatStrings, self.numTotalBat)

                self.SGG = HPS_PVGridGen(
                    self.PVPanel, self.numPV, self.Grid,
                    self.Inv, self.numInv, self.Gen,
                    self.numGen)

                self.SGBG = HPS_PVGridBatGen(
                    self.PVPanel, self.numPV, self.Grid, self.Inv,
                    self.numInv, self.Bat, self.numBatStrings,
                    self.numTotalBat, self.Gen, self.numGen)

In [31]: def checkObjectReplace(year, Object):
            if (year+1) % Object.warranty == 0:
                return True
            else:
                return False

In [32]: def economicalAnalysis(Config):
                opCost = [0]*years
                replaceCost = [0]*years
                totalCost       = [0]*years
                totalCost[0]    = Config.capitalCost*(
                    1+installationCost)
                Savings    = [0]*years

                cost    = 0
                savings = 0

                for i in range (0,years):
                    '''
```

```
                    Simulate for each year:
                        All values adjusted for inflation: assumed linear
                        Replacement costs: check warranty of system
                        Operational cost is a percentage of capital cost
                        Savings are based on what would for grid
                    '''
                    IR = math.pow((1 + inflation),i) #inflation rate

                    opCost[i] +=(Config.totalGridUsed*gridPrice \
                                 + Config.capitalCost*omPV)*IR
                    # PV replacement
                    if checkObjectReplace(i, Config.PVPanel) == True:
                        replaceCost[i]+=Config.PVPanel.Price*Config.numPV*IR
                    if checkObjectReplace(i, Config.Inv) == True:
                        replaceCost[i]+=Config.Inv.Price*Config.numInv*IR

                    # Bat
                    if 'Bat' in Config.Name:
                        opCost[i] += Config.capitalCost*omBat*IR

                        t=Config.batMatrix[0].dischargeEff*gridPrice*IR

                        Savings[i] += Config.totalBatUsed*t

                        if Config.checkBatReplace(i) == True:
                            replaceCost[i]+=Config.Bat.Price*Config.numBat*IR

                    # Gen
                    if 'Gen' in Config.Name:
                        opCost[i]+= Config.capitalCost*omGen*IR \
                                    + Config.totalFuelCost*IR
                        Savings[i]+=Config.totalGenUsed*gridPrice*IR

                    invEff = (Config.Inv.Eff/100)
                    Savings[i]+=Config.totalPVUsed*invEff*gridPrice*IR

                    totalCost[i]    += opCost[i] \
                                    + replaceCost[i]

                    cost    += totalCost[i]
                    savings += Savings[i]

                Config.IRR = savings/cost

In [33]: def setFitness(Config):
             wLPSP      = 12
             wPVUsage   = 4
             wGridUsage = 1
             wGenUsage  = 1
             wCF        = 3
             wROI       = 1
```

```
            # Lower = worse
            fitnessGridUsage = 1 - Config.GridUsage
            fitnessGenUsage = 1 - Config.GenUsage
            fitnessLPSP = 1 - Config.LPSP

            # if LPSP == 0: cannot divide by zero
            x = 0
            if Config.LPSP == 0:
                x = Config.capitalCost/0.01
            else:
                x = (Config.capitalCost/(Config.LPSP*100))

            # Higher = better
            fitnessCF = Config.CF
            fitnessIRR = Config.IRR
            fitnessPVUsage = Config.PVUsage

            Config.fitness =  wLPSP*fitnessLPSP \
                            + wPVUsage*fitnessPVUsage \
                            + wGridUsage*fitnessGridUsage \
                            + wGenUsage*fitnessGenUsage\
                            + wCF*fitnessCF \
                            + wROI*fitnessIRR

In [34]: def calcCapitalCost(Config):
            Config.capitalCost=Config.PVPanel.Price*Config.numPV \
                               + Config.Inv.Price*Config.numInv
            if 'Bat' in Config.Name:
                Config.capitalCost+=Config.Bat.Price*Config.numBat
            if 'Gen' in Config.Name:
                Config.capitalCost+=Config.Gen.Price*Config.numGen
```

C.1.8.2 HPS: PV-Grid

```
In [35]: class HPS_PVGrid:

            def __init__(self, PVPanel, numPV, Grid, Inv, numInv):
                # HPS Configuration
                self.Name = 'PV-Grid'
                self.PVPanel = PVPanel
                self.Grid = Grid
                self.numPV = numPV
                self.Inv = Inv
                self.numInv = numInv
                # Financial indicators
                self.capitalCost = 0
                self.IRR = 0
                # Fitness function
                self.fitness = 0
                # Technical and environmental indicators
                self.GenUsage = 0
                self.LPS_hourly=[0]*len(LoadDB)
```

```
        self.LPSP=0
        self.LPS_C=0
        self.LPS = 0
        self.CFHours=0
        self.totalLoad = 0
        self.totalPV = 0
        self.totalGrid = 0
        self.totalExcessPV = 0
        self.totalGridUsed = 0
        self.totalPVUsed = 0
        self.totalPVArea = self.PVPanel.area*self.numPV

        # Calculate total capital cost
        calcCapitalCost(self)
        # Calculate loss of power supply probability
        self.calcLPSP()

    def calcLPSP(self):
        '''
        Calculate loss of power supply probability.

        Control system constraints:
            1. Use all available PV
            2. Use grid if the load is not satisfied
        '''

        PV_eff  = (self.PVPanel.Eff/100)*derateFactor/1000
        Inv_eff = self.Inv.Eff/100

        for i in range (0, len(LoadDB)):
            PVUsed = 0
            PVExcess = 0
            gridUsed = 0

            # To calculate total load
            self.totalLoad += LoadDB[i].loadkW

            # reset loss of power supply
            LPS = LoadDB[i].loadkW

            # check if grid is available for hour
            gridAv  =  int(checkProb(
                self.Grid.reliable))*self.Grid.kWh
            self.totalGrid += gridAv

            # check if PV power is available
            PVAvailable  =  self.totalPVArea*(
                WeatherDB[i].insolation)*PV_eff
            self.totalPV += PVAvailable

            # if there is sufficient PV power
            # always use PV power first and as much as possible
```

```
                    if PVAvailable > LPS/Inv_eff:
                        PVUsed   = LPS/Inv_eff
                        PVExcess = PVAvailable - PVUsed
                        LPS = 0
                    # use whatever PV power is available
                    else:
                        PVUsed = PVAvailable
                        LPS -= PVUsed*Inv_eff

                    # If there is still a loss of power supply
                    if LPS > 0:
                        if LPS > gridAv:
                            LPS -= gridAv
                            gridUsed = gridAv
                        else:
                            gridUsed = LPS
                            LPS = 0

                    # If the load is completely supplied
                    LPS = max(0, LPS)
                    self.CFHours += int(LPS == 0)

                    # For analysis later
                    self.LPS_hourly[i] += LPS

                    # Analyse if this is part of critical hour analysis
                    if criticalHoursDB[WeatherDB[i].Time.hour] == True:
                        self.LPS_C += LPS

                    self.LPS += LPS

                    self.totalExcessPV += PVExcess
                    self.totalGridUsed += gridUsed
                    self.totalPVUsed += PVUsed

                self.LPSP   = self.LPS/self.totalLoad
                self.LPSP_C = self.LPS_C/self.totalLoad
                self.CF = self.CFHours/len(LoadDB)
                self.PVUsage  = self.totalPVUsed/self.totalPV
                self.GridUsage   = self.totalGridUsed/self.totalGrid
                self.ExcessPV = self.totalExcessPV/self.totalPV
```

C.1.8.3 HPS: PV-Grid-Battery

```
In [36]: class HPS_PVGridBat:

            def __init__(self, PVPanel, numPV, Grid, Inv, numInv,
                         Bat, batStrings, numBat):
                # HPS Configuration
                self.Name = 'PV-Grid-Bat'
                self.PVPanel=PVPanel
                self.Grid=Grid
```

```
        self.numPV=numPV
        self.Inv=Inv
        self.numInv=numInv
        self.Bat=Bat
        self.batStrings=batStrings
        self.numBat=numBat
        # Financial indicators
        self.capitalCost = 0
        self.IRR = 0
        # Fitness function
        self.fitness = 0
        # Technical and environmental indicators
        self.GenUsage = 0
        self.LPS_hourly=[0]*len(LoadDB)
        self.LPS_C=0
        self.CFHours=0
        self.totalLoad=0
        self.totalPV=0
        self.totalGrid=0
        self.totalExcessPV=0
        self.totalGridUsed=0
        self.totalPVUsed=0
        self.totalBatUsed=0
        self.totalPVArea=self.PVPanel.area*self.numPV
        self.LPS=0

        # Initialize batteries as a matrix: look at each bat
        # individually throughout 1 year
        self.initializeBatArray()
        # Calculate total capital cost
        calcCapitalCost(self)
        # Calculate loss of power supply
        self.calcLPSP()

    def calcLPSP(self):
        '''
        Calculate loss of power supply probability.

        Control system constraints:
        1. Use all available PV
        2. Discharge batteries
        3. Use grid if the load is not satisfied
        4. Charge batteries with excess PV and grid:
           get batteries to be as much charged as
           possible at all times.
        '''

        PV_eff  = self.PVPanel.Eff/100*derateFactor/1000
        Inv_eff = self.Inv.Eff/100

        for i in range (0, len(LoadDB)):
            PVUsed = 0
```

```
PVExcess = 0
gridUsed = 0
batUsed = 0

# To calculate total load
self.totalLoad += LoadDB[i].loadkW

# reset loss of power supply
LPS = LoadDB[i].loadkW

# check if grid is available for hour
gridAv  =  int(checkProb(
    self.Grid.reliable))*self.Grid.kWh
self.totalGrid += gridAv

# If it is a new month, update bat months
# for self-discharge efficiency
if i > 0:
    if LoadDB[i].Time.month != \
            LoadDB[i-1].Time.month:
        self.updateBatMonths()

# check if PV power is available
PVAvailable  =  self.totalPVArea*(
    WeatherDB[i].insolation)*PV_eff
self.totalPV += PVAvailable

# if there is sufficient PV power
if PVAvailable > LPS/Inv_eff:
    PVUsed   = LPS/Inv_eff
    PVExcess = PVAvailable - PVUsed
    LPS = 0
# use whatever PV power is available
else:
    PVUsed=PVAvailable
    LPS -= PVUsed*Inv_eff

# Batteries: discharge batt if there is still LPS
if LPS > 0:
    LPS, batUsed = self.dischargeBatMatrix(LPS)

# If there is still a loss of power supply
if LPS > 0:
    if LPS > gridAv:
        LPS -= gridAv
        gridUsed = gridAv
        gridAv = 0
    else:
        gridUsed = LPS
        gridAv -= LPS
        LPS = 0
```

```
                # charge bat if excessPV or grid power is available
                if (PVExcess > 0 or gridAv > 0) and LPS == 0:
                    s_used, g_used = self.chargeBatMatrix(
                        PVExcess, gridAv)
                    PVExcess -= s_used
                    PVUsed += s_used
                    gridUsed += g_used

                # If the load is completely supplied
                LPS = max(0, LPS)
                self.CFHours += int(LPS == 0)

                self.LPS_hourly[i] += LPS

                if criticalHoursDB[WeatherDB[i].Time.hour]== True:
                    self.LPS_C += LPS

                self.LPS += LPS
                self.totalExcessPV += PVExcess
                self.totalGridUsed += gridUsed
                self.totalPVUsed += PVUsed
                self.totalBatUsed += batUsed

        self.LPSP = self.LPS/self.totalLoad
        self.LPSP_C = self.LPS_C/self.totalLoad
        self.CF = self.CFHours/len(LoadDB)
        self.PVUsage = self.totalPVUsed/self.totalPV
        self.GridUsage = self.totalGridUsed/self.totalGrid
        self.ExcessPV = self.totalExcessPV/self.totalPV

    def initializeBatArray(self):
        '''
        Batteries are seen as an array.
        Each has own tracking of lifecycles, available power.
        '''
        self.batMatrix = []
        for i in range (0, self.numBat):
            self.batMatrix.append(BatObject(self.Bat))

    def chargeBatMatrix(self, PVExcess, gridAv):
        '''
        Charge the batteries using excess PV and available grid.

        This grid is only used if the batteries are less than
        40% of the total capacity.

        '''
        chargeEff = self.batMatrix[0].chargeEff

        PVUsed = 0
        gridUsed = 0
```

```
            for i in range (0, self.numBat):
                maxBat = self.batMatrix[i].maxBat
                if self.batMatrix[i].chargingState == True:
                    availableCapacity = self.batMatrix[i].maxBat \
                                        - self.batMatrix[i].minBat

                    if PVExcess > availableCapacity*chargeEff:
                        self.batMatrix[i].batPower = maxBat
                        PVUsed += availableCapacity/chargeEff
                        self.batMatrix[i].chargingState = False
                        PVExcess -= availableCapacity/chargeEff
                    else:
                        self.batMatrix[i].batPower+=PVExcess*chargeEff
                        PVUsed += PVExcess
                        PVExcess = 0

                if PVExcess <=0:
                    break

            totalCapacityAvailable=0
            totalCapacity = self.numBat*(
                self.batMatrix[0].maxBat - self.batMatrix[0].minBat)

            for i in range (0, self.numBat):
                totalCapacityAvailable += self.batMatrix[i].batPower \
                                          - self.batMatrix[i].minBat

            ratio=totalCapacityAvailable/totalCapacity
            if ratio < 0.4:
                maxBat = self.batMatrix[i].maxBat
                for i in range (0, self.numBat):
                    if self.batMatrix[i].chargingState == True:
                        availableCapacity = self.batMatrix[i].maxBat \
                                            - self.batMatrix[i].minBat

                        if gridAv > availableCapacity*chargeEff:
                            self.batMatrix[i].batPower = maxBat
                            gridUsed += availableCapacity/chargeEff
                            self.batMatrix[i].chargingState = False
                            gridAv -= availableCapacity/chargeEff
                        else:
                            self.batMatrix[i].batPower+=gridAv*chargeEff
                            gridUsed += gridAv
                            gridAv = 0
                            break
                    if gridAv <= 0:
                        break

            return PVUsed, gridUsed

        def dischargeBatMatrix(self, LPS):
            '''
```

```
        Discharge batteries based on the LPS value.

        Returns the leftover LPS and how much bat was discharged

        '''
        batUsed = 0
        battDischargeEff = self.batMatrix[0].dischargeEff
        for i in range (0, self.numBat):
            minBat = self.batMatrix[i].minBat
            selfDisch = self.batMatrix[i].totalSelfDischRate
            if self.batMatrix[i].chargingState == False:
                available = (self.batMatrix[i].batPower \
                             - self.batMatrix[i].minBat)*(1 \
                                    - selfDisch)*battDischargeEff

                if LPS > available:
                    LPS -= available
                    batUsed += (self.batMatrix[i].batPower \
                                - self.batMatrix[i].minBat)
                    self.batMatrix[i].batPower = minBat
                    self.batMatrix[i].chargingState = True
                    self.batMatrix[i].lifecycles += 1
                else:
                    used = LPS/((1-selfDisch)*battDischargeEff)
                    self.batMatrix[i].batPower -= used
                    LPS = 0
                    batUsed += used

                if LPS <= 0:
                    break

        return LPS, batUsed

    def updateBatMonths(self):
        '''
        Update months for self-discharge value
        '''
        for i in range (0, self.numBat):
            self.batMatrix[i].month+=1

    def checkBatReplace(self, year):
        '''
        Analyse individual bat life cycles and if it is finished.
        '''
        totalUsed=0
        totalAvailable=self.batMatrix[0].maxLifecycles*self.numBat
        for i in range (0, self.numBat):
            totalUsed+=self.batMatrix[i].lifecycles
        years_=math.floor(totalAvailable/totalUsed)
        if years_ > 0 and (year+1) % years_ == 0:
            return True
        else:
```

```
                    return False
```

```
In [37]: class BatObject:

             def __init__(self, Bat):
                 self.batPower = Bat.Power
                 self.month = 1
                 self.warranty = Bat.designLife
                 self.lifecycles = 0
                 self.maxLifecycles = 1800
                 self.chargingState = False
                 self.selfDischRate = 0.03
                 self.minBat = self.batPower*(1-DOD)
                 self.maxBat = self.batPower
                 self.totalSelfDischRate = self.month*self.selfDischRate
                 self.dischargeEff = 0.90
                 self.chargeEff = 0.90

             def __str__(self):
                 chargingState = ''
                 if self.chargingState == False:
                     chargingState = 'D'
                 else:
                     chargingState = 'C'
                 return ('%.1fkWh %d %s\t' % (self.batPower,
                                              self.lifecycles,
                                              chargingState))
```

C.1.8.4 HPS: PV-Grid-Generator

```
In [38]: class HPS_PVGridGen:

             def __init__(self, PVPanel, numPV, Grid, Inv, numInv,
                          Gen, numGen):
                 # HPS Configuration
                 self.Name = 'PV-Grid-Gen'
                 self.PVPanel=PVPanel
                 self.numPV=numPV
                 self.Grid=Grid
                 self.Inv=Inv
                 self.numInv=numInv
                 self.Gen=Gen
                 self.numGen=numGen
                 # Financial indicators
                 self.capitalCost = 0
                 self.IRR = 0
                 # Fitness function
                 self.fitness = 0
                 # Technical indicators
                 self.LPS_hourly=[0]*len(LoadDB)
                 self.LPS_C=0
                 self.LPS=0
```

```
        self.CFHours=0
        self.totalLoad=0
        self.totalPV=0
        self.totalGrid=0
        self.totalExcessPV=0
        self.totalGridUsed=0
        self.totalPVUsed=0
        self.totalGenUsed=0
        self.totalPVArea=self.PVPanel.area*self.numPV

        self.genMatrix=[GenObject(self.Gen) \
                        for x in range (self.numGen)]

        calcCapitalCost(self)
        self.calcLPSP()
        self.totalFuelCost=self.calcFuelCost()

    def calcLPSP(self):
        '''
        Calculate loss of power supply probability.

        Constraints:
        1. Use PV: charge batteries with excess
        2. Use running gens: charge batteries with excess
           (min load)
        3. Use grid
        '''

        PV_eff  = (self.PVPanel.Eff/100)*derateFactor/1000
        Inv_eff = self.Inv.Eff/100

        for i in range (0, len(LoadDB)):
            PVUsed = 0
            PVExcess = 0
            gridUsed = 0
            genUsed = 0

            # To calculate total load
            self.totalLoad+=LoadDB[i].loadkW

            # Rest LPS
            LPS=LoadDB[i].loadkW

            # check if grid is available for hour
            gridAv  =  int(checkProb(
                self.Grid.reliable))*self.Grid.kWh
            self.totalGrid += gridAv

            # Reset required gens
            for j in range (self.numGen):
                self.genMatrix[j].required = False
```

```
# check if PV power is available
PVAvailable = self.totalPVArea*(
    WeatherDB[i].insolation)*PV_eff
self.totalPV += PVAvailable

# if there is sufficient PV power
if PVAvailable > LoadDB[i].loadkW:
    PVUsed = LoadDB[i].loadkW/Inv_eff
    PVExcess = PVAvailable - PVUsed
    LPS=0
else:
    PVUsed = PVAvailable
    LPS -= PVUsed*Inv_eff

# check for gens that are already running and
# rather use these
for j in range (self.numGen):
    if self.genMatrix[j].checkRunning(
        minRunningHours) == True:
        self.genMatrix[j].switchOn()
        LPS-=self.Gen.Power
        if LPS < 0:
            LPS = 0


if LPS > gridAv and LPS !=0:
    #use gens
    for j in range (self.numGen):
        # Try full load first
        if LPS > self.Gen.Power:
            self.genMatrix[j].switchOn()
            LPS-=self.Gen.Power
            genUsed+=self.Gen.Power

        else:
            if LPS > gridAv:
                LPS-=gridAv
                gridUsed=gridAv
                gridAv=0
                break
            else:
                gridUsed=LPS
                gridAv-=LPS
                LPS=0
                break
else:
    gridUsed=LPS
    LPS=0

# Switch off any unused gens
for j in range (self.numGen):
    if self.genMatrix[j].required == False:
```

```
                        self.genMatrix[j].switchOff()

                LPS = max(0, LPS)
                self.CFHours += int(LPS == 0)

                self.LPS+=LPS
                self.LPS_hourly[i]+=LPS

                if criticalHoursDB[WeatherDB[i].Time.hour]== True:
                    self.LPS_C+=LPS

                self.totalGenUsed += genUsed
                self.totalExcessPV += PVExcess
                self.totalGridUsed += gridUsed
                self.totalPVUsed += PVUsed

            self.LPSP=self.LPS/self.totalLoad
            self.LPSP_C=self.LPS_C/self.totalLoad
            self.CF=self.CFHours/len(LoadDB)
            self.PVUsage=self.totalPVUsed/self.totalPV
            self.GridUsage=self.totalGridUsed/self.totalGrid
            self.ExcessPV=self.totalExcessPV/self.totalPV
            self.GenUsage=self.totalGenUsed/self.totalLoad

        def calcFuelCost(self):
            hours = 0
            fuel = 0
            for i in range (0, self.numGen):
                hours += self.genMatrix[i].runningHours
            fuel = self.Gen.fuel75Load*fuelPrice*hours
            return fuel

In [39]: class GenObject:

        def __init__(self, Gen):
            self.Gen=Gen
            self.runningHours=0
            self.switchOnHours=0
            self.switchedOn=False
            self.required=False

        def __str__(self):
            return ('%d\t%d\tRequired: %s' % (
                self.runningHours,
                self.switchOnHours, self.required))

        def checkRunning(self, minHours):
            if self.switchOnHours < minHours \
                        and self.switchOnHours != 0 \
                        and self.switchedOn == True:
                return True
            else:
```

```
                return False

        def switchOn(self):
            self.runningHours += 1
            self.switchOnHours += 1
            self.switchedOn = True
            self.required = True

        def switchOff(self):
            self.switchOnHours = 0
            self.switchedOn = False
```

C.1.8.5 HPS: PV-Grid-Battery-Generator

```
In [40]: class HPS_PVGridBatGen:

        def __init__(self, PVPanel, numPV, Grid, Inv, numInv, Bat,
                     batStrings, numBat, Gen, numGen):
            # HPS Configuration
            self.Name = 'PV-Grid-Bat-Gen'
            self.PVPanel=PVPanel
            self.Grid=Grid
            self.numPV=numPV
            self.Inv=Inv
            self.numInv=numInv
            self.Bat=Bat
            self.batStrings=batStrings
            self.numBat=numBat
            self.Gen       = Gen
            self.numGen    = numGen
            self.genMatrix = [GenObject(self.Gen) \
                              for x in range (self.numGen)]
            # Financial indicators
            self.capitalCost = 0
            self.IRR = 0
            # Fitness function
            self.fitness = 0
            # Technical and environmental indicators
            self.GenUsage = 0
            self.LPS_hourly = [0]*len(LoadDB)
            self.LPS_C = 0
            self.CFHours = 0
            self.totalLoad = 0
            self.totalPV = 0
            self.totalGrid = 0
            self.totalExcessPV = 0
            self.totalGridUsed = 0
            self.totalPVUsed = 0
            self.totalGenUsed = 0
            self.totalBatUsed = 0
            self.totalPVArea = self.PVPanel.area*self.numPV
            self.LPS = 0
```

```
        # Initialize batteries as a matrix: look at each bat
        # individually throughout 1 year
        self.initializeBatArray()
        # Calculate total capital cost
        calcCapitalCost(self)
        # Calculate loss of power supply
        self.calcLPSP()
        # Calculate total fuel cost
        self.totalFuelCost=self.calcFuelCost()

    def calcLPSP(self):
        '''
        Calculate loss of power supply probability.

        Control system constraints:
        1. Use all available PV
        2. Discharge batteries
        3. Use grid if the load is not satisfied
        4. Switch on gens if load is not satisfied
        5. Charge batteries with excess PV, grid and running
            gens (always run at 75%) to get batteries to be
            as much charged as possible at all times.
        '''

        PV_eff  = (self.PVPanel.Eff/100)*(derateFactor)/1000
        Inv_eff = self.Inv.Eff/100

        for i in range (0, len(LoadDB)):
            PVUsed = 0
            PVExcess = 0
            gridUsed = 0
            batUsed = 0
            genUsed = 0
            genExcess = 0

            # To calculate total load
            self.totalLoad += LoadDB[i].loadkW

            # reset loss of power supply
            LPS = LoadDB[i].loadkW

            # Reset gens
            for j in range (self.numGen):
                self.genMatrix[j].required=False

            # check if grid is available for hour
            gridAv  =  int(checkProb(
                self.Grid.reliable))*self.Grid.kWh
            self.totalGrid += gridAv

            # If it is a new month, update bat months
```

```
# for self-discharge efficiency
if i > 0:
    if LoadDB[i].Time.month != LoadDB[i-1].Time.month:
        self.updateBatMonths()

# check if PV power is available
PVAvailable  =  self.totalPVArea*(
    WeatherDB[i].insolation)*PV_eff
self.totalPV += PVAvailable

# if there is sufficient PV power
if PVAvailable > LPS/Inv_eff:
    PVUsed   = LPS/Inv_eff
    PVExcess = PVAvailable - PVUsed
    LPS = 0
else: # use whatever PV power is available
    PVUsed=PVAvailable
    LPS -= PVUsed*Inv_eff

#check for gens that are already running and
# rather use these
for j in range (self.numGen):
    if self.genMatrix[j].checkRunning(
        minRunningHours) == True:
        self.genMatrix[j].switchOn()
        LPS -= self.Gen.Power
        if LPS < 0:
            genExcess = np.absolute(LPS)
            LPS = 0

# Batteries: discharge bat if there is still a LPS
if LPS > 0:
    LPS, batUsed = self.dischargeBatMatrix(LPS)

# If there is still a loss of power supply
if LPS > 0:
    if LPS > gridAv:
        LPS -= gridAv
        gridUsed = gridAv
        gridAv = 0
    else:
        gridUsed = LPS
        gridAv -= LPS
        LPS = 0

# Switch on gens
if LPS > 0:
    for j in range (self.numGen):
        if LPS > 0:
            self.genMatrix[j].switchOn()
            LPS -= self.Gen.Power
            genUsed += self.Gen.Power
```

```
                    if LPS < 0:
                        genExcess = np.absolute(LPS)
                        LPS = 0
                        break

                # charge batteries if there is excess PV, grid power
                # available and whatever gen power is running
                # (do not switch on though - only if it has to run)
                if (PVExcess > 0 or gridAv > 0 or genExcess) \
                        and LPS <= 0:
                    s_used, g_used = self.chargeBatMatrix(
                        PVExcess, gridAv, genExcess)
                    PVExcess -= s_used
                    PVUsed += s_used
                    gridUsed += g_used

                # Switch off gens
                for j in range (self.numGen):
                    if self.genMatrix[j].required == False:
                        self.genMatrix[j].switchOff()

                # If the load is completely supplied
                LPS = max(0, LPS)
                self.CFHours += int(LPS == 0)

                self.LPS_hourly[i] += LPS

                if criticalHoursDB[WeatherDB[i].Time.hour]== True:
                    self.LPS_C += LPS

                self.LPS += LPS
                self.totalExcessPV += PVExcess
                self.totalGridUsed += gridUsed
                self.totalPVUsed += PVUsed
                self.totalBatUsed += batUsed
                self.totalGenUsed += genUsed

            self.LPSP = self.LPS/self.totalLoad
            self.LPSP_C = self.LPS_C/self.totalLoad
            self.CF = self.CFHours/len(LoadDB)
            self.PVUsage = self.totalPVUsed/self.totalPV
            self.GridUsage = self.totalGridUsed/self.totalGrid
            self.ExcessPV = self.totalExcessPV/self.totalPV
            self.GenUsage = self.totalGenUsed/self.totalLoad

        def initializeBatArray(self):
            '''
            Batteries are seen as an array.
            Each has own tracking of lifecycles, available power.
            '''
            self.batMatrix = []
            for i in range (0, self.numBat):
```

```
            self.batMatrix.append(BatObject(self.Bat))

    def chargeBatMatrix(self, PVExcess, gridAv, genExcess):
        '''
        Charge the batteries using excess PV and available grid.

        Always charge with PV and gen excess.

        This grid is only used if the batteries are less
        than 40% of the total capacity.

        '''
        chargeEff = self.batMatrix[0].chargeEff

        PVUsed = 0
        gridUsed = 0

        for i in range (0, self.numBat):
            minBat = self.batMatrix[i].minBat
            maxBat = self.batMatrix[i].maxBat
            if self.batMatrix[i].chargingState == True:
                availableCapacity = maxBat - minBat
                if PVExcess > availableCapacity*chargeEff:
                    self.batMatrix[i].batPower = maxBat
                    PVUsed += availableCapacity/chargeEff
                    self.batMatrix[i].chargingState = False
                    PVExcess -= availableCapacity/chargeEff
                else:
                    self.batMatrix[i].batPower+=PVExcess*chargeEff
                    PVUsed += PVExcess
                    PVExcess = 0
            if PVExcess <=0:
                break

        for i in range (0, self.numBat):
            if self.batMatrix[i].chargingState == True:
                availableCapacity = maxBat - minBat
                if genExcess > availableCapacity*chargeEff:
                    self.batMatrix[i].batPower = maxBat
                    self.batMatrix[i].chargingState = False
                    genExcess -= availableCapacity/chargeEff
                else:
                    self.batMatrix[i].batPower+=genExcess*chargeEff
                    genExcess = 0
            if genExcess <=0:
                break

        totalCapacityAvailable=0
        totalCapacity=self.numBat*(self.batMatrix[0].maxBat \
                                   - self.batMatrix[0].minBat)

        for i in range (0, self.numBat):
```

```
                totalCapacityAvailable += self.batMatrix[i].batPower \
                                          - self.batMatrix[i].minBat

            ratio=totalCapacityAvailable/totalCapacity

            if ratio < 0.4:
                for i in range (0, self.numBat):
                    maxBat = self.batMatrix[i].maxBat
                    if self.batMatrix[i].chargingState == True:
                        availableCapacity = self.batMatrix[i].maxBat \
                                            - self.batMatrix[i].minBat

                        if gridAv > availableCapacity*chargeEff:
                            self.batMatrix[i].batPower = maxBat
                            gridUsed += availableCapacity/chargeEff
                            self.batMatrix[i].chargingState=False
                            gridAv -= availableCapacity/chargeEff
                        else:
                            self.batMatrix[i].batPower+=gridAv*chargeEff
                            gridUsed += gridAv
                            gridAv = 0
                            break

                    if gridAv <= 0:
                        break

            return PVUsed, gridUsed

        def dischargeBatMatrix(self, LPS):
            '''
            Discharge batteries based on the LPS value.

            Returns the leftover LPS and how much bat was discharged

            '''
            batUsed = 0
            battDischargeEff = self.batMatrix[0].dischargeEff
            for i in range (0, self.numBat):
                minBat = self.batMatrix[i].minBat
                selfDisch = self.batMatrix[i].totalSelfDischRate
                if self.batMatrix[i].chargingState == False:
                    available = (self.batMatrix[i].batPower-minBat)*(
                        1 - selfDisch)*battDischargeEff
                    if LPS > available:
                        LPS -= available
                        batUsed += (self.batMatrix[i].batPower-minBat)
                        self.batMatrix[i].batPower = minBat
                        self.batMatrix[i].chargingState = True
                        self.batMatrix[i].lifecycles += 1
                    else:
                        used = LPS/((1-selfDisch)*battDischargeEff)
                        self.batMatrix[i].batPower -= used
```

```
                        LPS = 0
                        batUsed += used
                    if LPS <= 0:
                        break
            return LPS, batUsed

        def updateBatMonths(self):
            '''
            Update months for self-discharge value
            '''
            for i in range (0, self.numBat):
                self.batMatrix[i].month+=1

        def checkBatReplace(self, year):
            '''
            Analyse individual bat life cycles and if it is done.
            '''
            totalUsed=0
            totalAvailable=self.batMatrix[0].maxLifecycles*self.numBat
            for i in range (0, self.numBat):
                totalUsed += self.batMatrix[i].lifecycles
            years_ = math.floor(totalAvailable/totalUsed)
            if years_ > 0 and (year+1) % years_ == 0:
                return True
            else:
                return False

        def calcFuelCost(self):
            hours = 0
            fuel = 0
            for i in range (0, self.numGen):
                hours += self.genMatrix[i].runningHours
            fuel = self.Gen.fuel75Load*fuelPrice*hours
            return fuel
```

C.1.9 GA functions

C.1.9.1 Define population

```
In [41]: Population = []

        def returnType(length):
            return int(round((length-1)*random.random()))

        def returnNumber(Min, Max):
            return int(round(Min + (Max-Min)*random.random()))

        def definePopulationChromosomes():
            for c in range(0, population):
                Population.append(Chromosome())
                Population[c].setTypes(
                    len(PVDB), len(BatDB),
```

```
                    len(GenDB), len(InvDB))
                Population[c].setClasses(
                    PVDB[Population[c].typePV],
                    BatDB[Population[c].typeBat],
                    GenDB[Population[c].typeGen],
                    InvDB[Population[c].typeInv],
                    grid)
                Population[c].setSizes(
                    SystemVolt, minBatStrings,
                    maxBatStrings, minGen,
                    maxGen, minInv, maxInv)

        definePopulationChromosomes()
```

C.1.9.2 Objective Functions

```
In [42]: def checkProb(prob):
             t = random.random()
             if t <= prob:
                 return True
             else:
                 return False

         def setPopulationHPSConfigurations():
             for i in range (0, population):
                 Population[i].setHPS()
                 Population[i].EconomicalAnalysis()

         setPopulationHPSConfigurations()
```

```
In [43]: def setRouletteWheel(popList):
             totalFitness = 0
             for i in range (0, len(popList)):
                 totalFitness += popList[i].fitness
             for i in range (0, len(popList)):
                 length=popList[i].fitness/totalFitness
                 if i == 0:
                     popList[i].startWheel = 0
                     popList[i].endWheel  = length
                 else:
                     popList[i].startWheel=popList[i-1].endWheel
                     popList[i].endWheel=popList[i].startWheel+length

         def returnDNAMutations(DNA_MutationProb):
             m = []
             for i in range (0, 7):
                 m.append(checkProb(DNA_MutationProb))
             return m


         def returnWeakestIndex(popList):
             weakest = popList[0]
```

```
        n = 0
        for i in range (0, len(popList)):
            if popList[i].fitness < weakest.fitness and i > 0:
                weakest = popList[i]
                n = i
        return n

    def returnTopFittest(listPop, maxFit):
        popList = listPop[:]
        for passnum in range (len(popList)-1,0,-1):
            for i in range (passnum):
                if popList[i].fitness <popList[i+1].fitness:
                    temp         = popList[i]
                    popList[i]   = popList[i+1]
                    popList[i+1] = temp
        returnList = []
        for i in range (0,maxFit):
            returnList.append(popList[i])
        return returnList

    def returnFittestSG(listPop):
        popList=listPop[:]
        for passnum in range (len(popList)-1,0,-1):
            for i in range (passnum):
                if popList[i].SG.fitness<popList[i+1].SG.fitness:
                    temp         = popList[i]
                    popList[i]   = popList[i+1]
                    popList[i+1] = temp
        return popList[0].SG

    def returnFittestSGB(listPop):
        popList = listPop[:]
        for passnum in range (len(popList)-1,0,-1):
            for i in range (passnum):
                if popList[i].SGB.fitness<popList[i+1].SGB.fitness:
                    temp         = popList[i]
                    popList[i]   = popList[i+1]
                    popList[i+1] = temp
        return popList[0].SGB

    def returnFittestSGG(listPop):
        popList = listPop[:]
        for passnum in range (len(popList)-1,0,-1):
            for i in range (passnum):
                if popList[i].SGG.fitness<popList[i+1].SGG.fitness:
                    temp         = popList[i]
                    popList[i]   = popList[i+1]
                    popList[i+1] = temp
        return popList[0].SGG

    def returnFittestSGBG(listPop):
        popList = listPop[:]
```

```
            for passnum in range (len(popList)-1,0,-1):
                for i in range (passnum):
                    if popList[i].SGBG.fitness<popList[i+1].SGBG.fitness:
                        temp = popList[i]
                        popList[i] = popList[i+1]
                        popList[i+1] = temp
            return popList[0].SGBG
```

C.1.9.3 Fitness Function

```
In [44]: def setPopulationFitness():
             for i in range (0, len(Population)):
                 Population[i].setFitness()

         setPopulationFitness()
```

C.1.9.4 Roulette Wheel

```
In [45]: setRouletteWheel(Population)
```

C.1.9.5 Offspring generation

```
In [46]: def findParent(probIndex, popList):
             parent = 0
             for i in range (0, len(popList)):
                 if popList[i].startWheel < probIndex \
                     and popList[i].endWheel >= probIndex:
                     parent = i
                     break
             return parent


         def returnPValue(parent1, parent2):
             if checkProb(0.5) == True:
                 return parent1
             else:
                 return parent2
```

C.1.9.6 Crossover

```
In [47]: def crossover():
             coProb1 = random.uniform(0,1)
             P1 = findParent(coProb1, Population)
             coProb2 = random.uniform(0,1)
             P2 = findParent(coProb2, Population)

             while P1 == P2:
                 P2 = random.uniform(0,1)
                 P2 = findParent(P2, Population)

             TPV = returnPValue(Population[P1].typePV,
                                Population[P2].typePV)
```

```
            TInv = returnPValue(Population[P1].typeInv,
                                Population[P2].typeInv)

            TBat = returnPValue(Population[P1].typeBat,
                                Population[P2].typeBat)

            TGen = returnPValue(Population[P1].typeGen,
                                Population[P2].typeGen)

            NInv = returnPValue(Population[P1].numInv,
                                Population[P2].numInv)

            NBatS = returnPValue(Population[P1].numBatStrings,
                                 Population[P2].numBatStrings)

            NGen = returnPValue(Population[P1].numGen,
                                Population[P2].numGen)

            generateOffspring(TPV, TBat, TGen, TInv, NBatS,
                              NGen, NInv)
```

C.1.9.7 Mutation

```
In [48]: def mutation():
            mutProb = random.uniform(0,1)
            Parent = findParent(mutProb, Population)
            mut = returnDNAMutations(DNA_MutationProb)
            # PV
            TPV=int(mut[0] == False)*Population[Parent].typePV \
                + int(mut[0] == True)*returnType(len(PVDB))
            # Inv
            TInv=int(mut[3] == False)*Population[Parent].typeInv \
                + int(mut[3] == True)*returnType(len(InvDB))
            NInv=int(mut[6] == False)*Population[Parent].numInv \
                + int(mut[6] == True)*returnNumber(minInv,maxInv)
            # Bat
            TBat=int(mut[1]==False)*Population[Parent].typeBat \
                + int(mut[1]==True)*returnType(len(BatDB))
            NBatS=int(mut[4]==False)*Population[Parent].numBatStrings \
                + int(mut[4]==True)*returnNumber(
                minBatStrings,maxBatStrings)
            # Gen
            TGen=int(mut[2] == False)*Population[Parent].typeGen \
                + int(mut[2] == True)*returnType(len(GenDB))
            NGen=int(mut[5] == False)*Population[Parent].numGen \
                + int(mut[5] == True)*returnNumber(minGen,maxGen)

            generateOffspring(TPV, TBat, TGen, TInv, NBatS, NGen, NInv)
```

C.1.9.8 Offspring

```
In [49]: def generateOffspring (TPV, TBat, TGen, TInv,
                                NBatS, NGen, NInv):
             offspring=Chromosome()
             offspring.setOffspring(TPV, TBat, TGen, TInv,
                                    NBatS, NGen, NInv)
             offspring.setClasses(PVDB[offspring.typePV],
                                  BatDB[offspring.typeBat],
                                  GenDB[offspring.typeGen],
                                  InvDB[offspring.typeInv],
                                  grid)
             offspring.setBatSize(SystemVolt)
             offspring.setPVSize()
             offspring.setHPS()
             offspring.EconomicalAnalysis()
             offspring.setFitness()
             weakestIndex=returnWeakestIndex(Population)
             if Population[weakestIndex].fitness < offspring.fitness:
                 global offspringCount
                 offspringCount += 1
                 Population[weakestIndex] = offspring
             setRouletteWheel(Population)
```

C.1.10 Design of HPS using GA

```
In [50]: def calcAvgFitness():
             totalFitness = 0
             for i in range (0, len(Population)):
                 totalFitness += Population[i].fitness
             avgFit = totalFitness/len(Population)
             return avgFit

         avgFitGen=[0]*generations


In [51]: top_fit = [0]*generations
         top_configs = [0]*generations

         def geneticAlgorithm():
             SG   = returnFittestSG(Population)
             SGB  = returnFittestSGB(Population)
             SGG  = returnFittestSGG(Population)
             SGBG = returnFittestSGBG(Population)

             count = 0

             for gen in range(0,generations):
                 SG_   = returnFittestSG(Population)
                 SGB_  = returnFittestSGB(Population)
                 SGG_  = returnFittestSGG(Population)
                 SGBG_ = returnFittestSGBG(Population)
                 if SG_ != SG:
```

```
                    SG = SG_
                if SGB_ != SGB:
                    SG = SGB_
                if SGG_ != SGG:
                    SGG = SGG_
                if SGBG_ != SGBG:
                    SGBG = SGBG_
                top_configs[gen] = SG.fitness + SGB.fitness + SGG.fitness + SGBG.fitness
                if gen >=1:
                    if top_configs[gen-1] != top_configs[gen]:
                        count+=1
                avgFitGen[gen] = calcAvgFitness()
                if checkProb(crossoverProb) == True:
                    crossover()
                else:
                    mutation()

            print('Total changes: {}\t{}%'.format(count, 100*count/generations))
            global offspringCount
            print('New Offsprings: {}'.format(offspringCount))

        geneticAlgorithm()
```

```
Total changes: 8        10.0%
New Offsprings: 71
```

C.1.11 Results

```
In [53]: SG=returnFittestSG(Population)
         SGB=returnFittestSGB(Population)
         SGG=returnFittestSGG(Population)
         SGBG=returnFittestSGBG(Population)
```

```
In [54]: # Write to table for .tex file
         config_DF=pd.DataFrame()

         SG_data={'Configuration':'SG',
                  'LPSP': '{:.1f}'.format(SG.LPSP*100),
                  'LPSP_C': '{:.1f}'.format(SG.LPSP_C*100),
                  'Capital Cost': 'R{:,.0f}'.format(SG.capitalCost),
                  'ROI': '{:.1f}'.format(SG.IRR*100),
                  'PV Utilisation': '{:.1f}'.format(SG.PVUsage*100),
                  'Gen Utilisation': '{:.1f}'.format(SG.GenUsage*100),
                  'Capacity Factor': '{:.1f}'.format(SG.CF*100)}

         SGB_data={'Configuration':'SGB',
                  'LPSP': '{:.1f}'.format(SGB.LPSP*100),
                  'LPSP_C': '{:.1f}'.format(SGB.LPSP_C*100),
                  'Capital Cost': 'R{:,.0f}'.format(SGB.capitalCost),
                  'ROI': '{:.1f}'.format(SGB.IRR*100),
```

```
            'PV Utilisation': '{:.1f}'.format(SGB.PVUsage*100),
            'Gen Utilisation': '{:.1f}'.format(SGB.GenUsage*100),
            'Capacity Factor': '{:.1f}'.format(SGB.CF*100)}

SGG_data={'Configuration':'SGG',
            'LPSP': '{:.1f}'.format(SGG.LPSP*100),
            'LPSP_C': '{:.1f}'.format(SGG.LPSP_C*100),
            'Capital Cost': 'R{:,.0f}'.format(SGG.capitalCost),
            'ROI': '{:.1f}'.format(SGG.IRR*100),
            'PV Utilisation': '{:.1f}'.format(SGG.PVUsage*100),
            'Gen Utilisation': '{:.1f}'.format(SGG.GenUsage*100),
            'Capacity Factor': '{:.1f}'.format(SGG.CF*100)}

SGBG_data={'Configuration':'SGBG',
            'LPSP': '{:.2f}'.format(SGBG.LPSP*100),
            'LPSP_C': '{:.2f}'.format(SGBG.LPSP_C*100),
            'Capital Cost': 'R{:,.0f}'.format(SGBG.capitalCost),
            'ROI': '{:.1f}'.format(SGBG.IRR*100),
            'PV Utilisation': '{:.1f}'.format(SGBG.PVUsage*100),
            'Gen Utilisation': '{:.1f}'.format(SGBG.GenUsage*100),
            'Capacity Factor': '{:.1f}'.format(SGBG.CF*100)}

config_DF=config_DF.append(SG_data, ignore_index=True)
config_DF=config_DF.append(SGB_data, ignore_index=True)
config_DF=config_DF.append(SGG_data, ignore_index=True)
config_DF=config_DF.append(SGBG_data, ignore_index=True)

config_DF
```

Out[54]:

	Capacity Factor	Capital Cost	Configuration	Gen Utilisation	LPSP	LPSP_C	PV Utilisation	ROI
0	35.3	R3,051,177	SG	0.0	42.1	26.9	47.5	23.4
1	72.7	R5,335,826	SGB	0.0	23.9	13.4	64.5	26.2
2	52.3	R2,405,780	SGG	40.9	8.3	5.3	66.5	19.0
3	100.0	R4,400,192	SGBG	37.6	0.01	0.01	91.1	18.3

In [55]:

```
configComponents_DF=pd.DataFrame(
    columns=['Configuration','Name', 'QTY', 'Power'])

SG_1={'Configuration':'SG',
        'Name': SG.PVPanel.Name,
        'QTY': SG.numPV,
        'Power': 'Peak Power {:.1f} kW'.format(
            SG.numPV*SG.PVPanel.Power/1000)}

SG_2={'Configuration':'SG',
        'Name': SG.Inv.Name,
```

```
        'QTY': SG.numInv}

SGB_1={'Configuration':'SGB',
        'Name': SGB.PVPanel.Name,
        'QTY': SGB.numPV,
        'Power': 'Peak Power {:.1f} kW'.format(
            SGB.numPV*SGB.PVPanel.Power/1000)}

SGB_2={'Configuration':'SGB',
        'Name': SGB.Inv.Name,
        'QTY': SGB.numInv}

SGB_3={'Configuration':'SGB',
        'Name': SGB.Bat.Name,
        'QTY': SGB.numBat,
        'Power': 'Storage {:.1f} kWh'.format(
            SGB.numBat*SGB.Bat.Power)}

SGG_1={'Configuration':'SGG',
        'Name': SGG.PVPanel.Name,
        'QTY': SGG.numPV,
        'Power': 'Peak Power {:.1f} kW'.format(
            SGG.numPV*SGG.PVPanel.Power/1000)}

SGG_2={'Configuration':'SGG',
        'Name': SGG.Inv.Name,
        'QTY': SGG.numInv}

SGG_3={'Configuration':'SGG',
        'Name': SGG.Gen.Name,
        'QTY': SGG.numGen,
        'Power': '{:.1f} kVA'.format(
            SGG.numGen*SGG.Gen.kVA)}

SGBG_1={'Configuration':'SGBG',
        'Name': SGBG.PVPanel.Name,
        'QTY': SGBG.numPV,
        'Power': 'Peak Power {:.1f} kW'.format(
            SGBG.numPV*SGBG.PVPanel.Power/1000)}

SGBG_2={'Configuration':'SGBG',
        'Name': SGBG.Inv.Name,
        'QTY': SGBG.numInv}

SGBG_3={'Configuration':'SGBG',
        'Name': SGBG.Gen.Name,
        'QTY': SGBG.numGen,
        'Power': '{:.1f} kVA'.format(
            SGBG.numGen*SGBG.Gen.kVA)}

SGBG_4={'Configuration':'SGBG',
        'Name': SGBG.Bat.Name,
```

```
            'QTY': SGBG.numBat,
            'Power': 'Storage {:.1f} kWh'.format(
                SGBG.numBat*SGBG.Bat.Power)}

        configComponents_DF=configComponents_DF.append(
            SG_1, ignore_index=True)
        configComponents_DF=configComponents_DF.append(
            SG_2, ignore_index=True)
        configComponents_DF=configComponents_DF.append(
            SGB_1, ignore_index=True)
        configComponents_DF=configComponents_DF.append(
            SGB_2, ignore_index=True)
        configComponents_DF=configComponents_DF.append(
            SGB_3, ignore_index=True)
        configComponents_DF=configComponents_DF.append(
            SGG_1, ignore_index=True)
        configComponents_DF=configComponents_DF.append(
            SGG_2, ignore_index=True)
        configComponents_DF=configComponents_DF.append(
            SGG_3, ignore_index=True)
        configComponents_DF=configComponents_DF.append(
            SGBG_1, ignore_index=True)
        configComponents_DF=configComponents_DF.append(
            SGBG_2, ignore_index=True)
        configComponents_DF=configComponents_DF.append(
            SGBG_3, ignore_index=True)
        configComponents_DF=configComponents_DF.append(
            SGBG_4, ignore_index=True)

        configComponents_DF
```

Out[55]:

	Configuration	Name	QTY	Power
0	SG	Canadian Solar 395W	551	Peak Power 217.6 kW
1	SG	Sunny Tripower 15000TL	29	NaN
2	SGB	Canadian Solar 395W	684	Peak Power 270.2 kW
3	SGB	Sunny Tripower 25000TL	36	NaN
4	SGB	OmniPower 240Ah 12V	190	Storage 547.2 kWh
5	SGG	Canadian Solar 395W	323	Peak Power 127.6 kW
6	SGG	Sunny Tripower 20000TL	19	NaN
7	SGG	MAC AFRIC 34 kVA	3	102.0 kVA
8	SGBG	Renewsys Galactic 365W	312	Peak Power 113.9 kW
9	SGBG	Sunny Tripower 25000TL	26	NaN
10	SGBG	MAC AFRIC 50 kVA	3	150.0 kVA
11	SGBG	OmniPower 180Ah 12V	190	Storage 410.4 kWh

C.2 Yr.no Data scraping

C.2.1 Imports

In [1]:
```
from yr.libyr import Yr
import pandas as pd
```

```
import datetime as dt
from time import sleep
import os
```

C.2.2 Site information

```
In [2]: latitude=-33.9346
        longitude=18.8668
        altitude=122
        file_path='C:/Users/18296033/Documents/WORK/DataCollector/datapoints'
```

C.2.3 Clean data

```
In [3]: def cleanUp (df, timestamp):
            for index, row in df.iterrows():
                if row['@from'] != row['@to']:
                    df=df.drop(index)

            df['@from']=pd.to_datetime(df['@from'])
            df['@to']=pd.to_datetime(df['@to'])

            df['Timestamp']=''

            for index, row in df.iterrows():
                if row['@from'] < timestamp and row['@from'].hour != timestamp.hour:
                    #print (row['@from'], timestamp)
                    df=df.drop(index)
                else:
                    value=str(timestamp.strftime('%Y-%m-%d %H:%M'))
                    df.at[index,'Timestamp']=value

            df=pd.concat([df.drop(['location'],
                                  axis=1),
                          df['location'].apply(pd.Series)],
                         axis=1)
            df=pd.concat([df.drop(['temperature'],
                                  axis=1),
                          df['temperature'].apply(pd.Series)],
                         axis=1)
            df=df.drop(['@id','@unit'],
                       axis=1)
            df=pd.concat([df.drop(['windDirection'],
                                  axis=1),
                          df['windDirection'].apply(pd.Series)],
                         axis=1)
            df=df.drop(['@id','@name'],
                       axis=1)
            df=df.rename(index=str,
                         columns={'@datatype':'Datatype',
                                  '@from':'From',
                                  '@to':'To',
                                  '@altitude':'Altitude',
```

```
                        '@latitude':'Latitude',
                        '@longitude':'Longitude',
                        '@value':'Temperature',
                        '@deg':'Degrees'
                       })
df=pd.concat([df.drop(['windSpeed'],
                        axis=1),
              df['windSpeed'].apply(pd.Series)],
             axis=1)
df=df.drop(['@id','@name','@beaufort'],axis=1)
df=df.rename(index=str,
             columns={'@mps':'Windspeed'})
df=pd.concat([df.drop(['humidity'],
                        axis=1),
              df['humidity'].apply(pd.Series)],
             axis=1)
df=df.drop(['@unit'],
            axis=1)
df=df.rename(index=str,
             columns={'@value':'Humidity'})
df=pd.concat([df.drop(['pressure'],
                        axis=1),
              df['pressure'].apply(pd.Series)],
             axis=1)
df=df.drop(['@unit','@id'],
            axis=1)
df=df.rename(index=str,
             columns={'@value':'Pressure'})
df=pd.concat([df.drop(['cloudiness'],
                        axis=1),
              df['cloudiness'].apply(pd.Series)],
             axis=1)
df=df.drop(['@id'],
            axis=1)
df=df.rename(index=str,
             columns={'@percent':'Clouds'})
df=pd.concat([df.drop(['fog'],
                        axis=1),
              df['fog'].apply(pd.Series)],
             axis=1)
df=df.drop(['@id'],
            axis=1)
df=df.rename(index=str,
             columns={'@percent':'Fog'})
df=pd.concat([df.drop(['lowClouds'],
                        axis=1),
              df['lowClouds'].apply(pd.Series)],
             axis=1)
df=df.drop(['@id'],
            axis=1)
df=df.rename(index=str,
             columns={'@percent':'Low Clouds'})
```

```
            df=pd.concat([df.drop(['mediumClouds'],
                                  axis=1),
                          df['mediumClouds'].apply(pd.Series)],
                         axis=1)
            df=df.drop(['@id'],
                       axis=1)
            df=df.rename(index=str,
                         columns={'@percent':'Medium Clouds'})
            df=pd.concat([df.drop(['highClouds'],
                                  axis=1),
                          df['highClouds'].apply(pd.Series)],
                         axis=1)
            df=df.drop(['@id'],
                       axis=1)
            df=df.rename(index=str,
                         columns={'@percent':'High Clouds'})
            df=pd.concat([df.drop(['dewpointTemperature'],
                                  axis=1),
                          df['dewpointTemperature'].apply(pd.Series)],
                         axis=1)
            df=df.drop(['@unit','@id'],axis=1)
            df=df.rename(index=str,
                         columns={'@value':'Dew Temperature'})
            return df
```

C.2.4 Collect data

```
In [ ]: while True:
            time=dt.datetime.now()

            if time.minute==0:
                weather = Yr(location_xyz=(longitude, latitude, altitude))
                df=pd.DataFrame.from_dict(weather.forecast())
                df=cleanUp(df,time)

                #write to hourly csv file
                file_name=str(time.strftime('%d_%m_%y %Hh%Mm%S.csv'))
                path=os.path.join(file_path, file_name)
                df.to_csv(path)
                print(file_name,'saved')

                #update weatherdatabase csv file
                newpath='C:/Users/18296033/Documents/WORK/DataCollector'
                newfilename='WeatherDataBase.csv'
                p=os.path.join(newpath, newfilename)
                oldDF=pd.read_csv(p)
                oldDF=oldDF.drop(['Unnamed: 0'], axis=1)
                oldDF=oldDF.append(df)
                os.remove(p)
                oldDF.to_csv(p)
                #sleep to the next hour
                sleep(3500)
```

C.3 Linear Regression

This regression technique is developed from a workshop tutorial on a basic regression with Keras using TensorFlow [73].

C.3.1 Imports

```
In [1]: import pandas as pd
        import glob

        import numpy as np
        import os

        import pvlib
        from pvlib.location import Location, solarposition

        import tensorflow as tf
        from tensorflow import keras
        from tensorflow.keras import layers

        import matplotlib.pyplot as plt
```

C.3.2 Graph Plot Variables

```
In [2]: label = 20
        ticklabel = 18

        params = {'axes.labelsize': label,
                  'axes.titlesize': label,
                  'text.usetex'   : True,
                  'font.size': ticklabel,
                  'legend.frameon': True,
                  'legend.fontsize': ticklabel,
                  'legend.loc': 'center right',
                  'legend.framealpha': 1,
                  'xtick.labelsize': ticklabel,
                  'ytick.labelsize': ticklabel,
                  'grid.linestyle' : ":",
                  'axes.grid': True,
                  'axes.grid.which': 'major',
                  'axes.grid'      : True,
                  'grid.linewidth' : 0.8,
                  'keymap.grid': ['g'],
                  'keymap.grid_minor': ['G'],
                  'figure.figsize': [18, 8]
                 }
        plt.rcParams.update(params)
```

C.3.3 Prediction Data

```
In [3]: tiltAngle=30
        rho=0.2
        LTM=-30
        latitude=-33.9346
        longitude=18.8668
        altitude=122

        loc=Location(latitude,
                     longitude,
                     'Africa/Johannesburg',
                     altitude,
                     'Stellenbosch')
```

```
In [4]: newpath='C:/Users/18296033/Documents/WORK/DataCollector'
        all_files = glob.glob(path + "/*.csv")

        li = pd.DataFrame()

        for filename in all_files:
            df = pd.read_csv(filename, index_col=None, header=0)
            li=li.append(df)
```

```
In [5]: li = li.drop(['Altitude', 'Latitude',
                    'Longitude', 'Datatype', 'To'], axis=1)
```

```
In [6]: li = li.drop(['0', '0.1', '0.2', 'Unnamed: 0'], axis=1)
```

```
In [7]: df = li.copy()

        df['From'] = pd.to_datetime(df['From'])
        df['Timestamp'] = pd.to_datetime(df['Timestamp'])
```

```
In [8]: result=df.sort_values(by=['From','Timestamp'],
                              ascending=[True, True])
```

```
In [9]: timestamp=result[result['From'] == result['Timestamp']]
```

```
In [10]: df_Predictions = timestamp.copy()
         df_Predictions['Date'] = df_Predictions['From']
         df_Predictions = df_Predictions.drop(['From', 'Timestamp'], axis=1)
```

C.3.4 Normal sky data

```
In [11]: df_Actual = pd.read_csv('sb_thour.csv')

         df_Actual=df_Actual.drop(['Record',
                                   'Date',
                                   'Time',
```

```
                                    'UVA_Avg',
                                    'UVB_Avg',
                                    'Tracker2WM_Avg',
                                    'WindDir_D1_WVT',
                                    'WindDir_SD1_WVT',
                                    'ShadowbandWM_Avg',
                                    'AirTC_Max',
                                    'DNICalc_Avg',
                                    'BattV_Min',
                                    'AirTC_Min'],
                                   axis=1)

        df_Actual=df_Actual.rename(index=str,
                                   columns={'SunWM_Avg':'GHI',
                                            'RH':'Humidity',
                                            'AirTC_Avg':'Temperature',
                                            'WS_ms_S_WVT':'Windspeed',
                                            'BP_mB_Avg':'Pressure',
                                            'TrackerWM_Avg':'DNI',
                                            'ShadowWM_Avg':'DHI',
                                            'TimeStamp':'Date'})

        df_Actual['Date']=pd.to_datetime(df_Actual['Date'])

        df_Actual['Month']=pd.DatetimeIndex(df_Actual['Date']).month
        df_Actual['Day']=pd.DatetimeIndex(df_Actual['Date']).day
        df_Actual['Day of year']=pd.DatetimeIndex(df_Actual['Date']).dayofyear
```

In [12]:
```
df_Actual=df_Actual.reset_index(drop=True)
df_Actual.shape
print(df_Actual.shape)
```

```
(12349, 11)
```

In [13]:
```
def getTotalIrradiance(GHI, DNI, DHI, timestamp):
    timestamp=pd.DatetimeIndex(timestamp)
    timestamp=timestamp.tz_localize(loc.tz)

    model='isotropic'

    # Panel azimuth from North
    surface_azimuth=0

    albedo=rho #surface albedo - declared in variables and constants

    # convert to radians
    latitudeRad=np.deg2rad(latitude)
    n=timestamp.dayofyear
    # Equation of time
    eqOfTime=pvlib.solarposition.equation_of_time_pvcdrom(n)
```

```
        # Hour angle
        hourangle=pvlib.solarposition.hour_angle(timestamp,
                                                 longitude,
                                                 eqOfTime)
        hourAngleRad=np.deg2rad(hourangle)

        #Declination angle calculations
        declinationRad=pvlib.solarposition.declination_cooper69(n)

        solar_zenithRad=pvlib.solarposition.solar_zenith_analytical(latitudeRad,
                                                                    hourAngleRad,
                                                                    declinationRad)

        #Solar azimuth angle calculations
        solar_azimuthRad=pvlib.solarposition.solar_azimuth_analytical(latitudeRad,
                                                                      hourAngleRad,
                                                                      declinationRad,
                                                                      solar_zenithRad)
        #convert to degrees
        solar_azimuth = np.rad2deg(solar_azimuthRad)
        solar_zenith= np.rad2deg(solar_zenithRad)
        surface_tilt=tiltAngle
        # get total irradiance
        total=pvlib.irradiance.get_total_irradiance(surface_tilt=tiltAngle,
                                                    surface_azimuth=surface_azimuth,
                                                    solar_zenith=solar_zenith,
                                                    solar_azimuth=solar_azimuth,
                                                    dni=DNI,
                                                    ghi=GHI,
                                                    dhi=DHI,
                                                    albedo=albedo,
                                                    model=model)

        return total

    total=getTotalIrradiance(df_Actual['GHI'],
                             df_Actual['DNI'],
                             df_Actual['DHI'],
                             pd.DatetimeIndex(df_Actual['Date']))

    df_Actual['Global Insolation']=total['poa_global']
    df_Actual['Direct Insolation']=total['poa_direct']
    df_Actual['Diffuse Insolation']=total['poa_diffuse']
    df_Actual['Sky Diffuse Insolation']=total['poa_sky_diffuse']
    df_Actual['Ground Insolation']=total['poa_ground_diffuse']
```

C.3.4.1 Add previous hour to dataset

```
In [14]: for i in range(1, len(df_Actual)):
             df_Actual.loc[i, 'Prev NS'] = df_Actual.loc[i-1,
                                                         'Global Insolation']
```

C.3.5 Clear sky model

```
In [15]: start=df_Predictions.iloc[0]['Date']
         end=df_Predictions.iloc[-1]['Date']

         timesClearSky=pd.date_range(start, end, freq='1h',
                             tz=loc.tz)

         df_Clearsky=loc.get_clearsky(timesClearSky)
         df_Clearsky['Date']=df_Clearsky.index.get_values()
         df_Clearsky['Date']=pd.to_datetime(
             df_Clearsky['Date']) + pd.to_timedelta(2,unit='h')

         df_Clearsky=df_Clearsky.rename(index=str,
                                        columns={'ghi':'GHI',
                                                 'dni':'DNI',
                                                 'dhi':'DHI'})

         df_Clearsky=df_Clearsky.reset_index(drop=True)

         total=getTotalIrradiance(df_Clearsky['GHI'],
                                  df_Clearsky['DNI'],
                                  df_Clearsky['DHI'],
                                  pd.DatetimeIndex(
                                      df_Clearsky['Date']))

         df_Clearsky['Global Insolation']=total['poa_global']
         df_Clearsky['Direct Insolation']=total['poa_direct']
         df_Clearsky['Diffuse Insolation']=total['poa_diffuse']
         df_Clearsky['Sky Diffuse Insolation']=total['poa_sky_diffuse']
         df_Clearsky['Ground Insolation']=total['poa_ground_diffuse']
```

C.3.6 Correlation Matrix

```
In [16]: corrMatrix_Actual=df_Actual.drop(['Pressure','Month',
                                         'Direct Insolation', 'Day',
                                               'Day of year', 'Prev NS'],
                                            axis=1).corr(method='pearson')
         corrMatrix_Actual.style.background_gradient(
             cmap='coolwarm').set_precision(3)
```

```
In [17]: corrMatrix_Pred=df_Predictions.drop(['Fog','Pressure',
                                              'Degrees', 'Dew Temperature'],
                                             axis=1).corr(method='pearson')
         corrMatrix_Pred.style.background_gradient(
             cmap='coolwarm').set_precision(3)
```

C.3.7 Merge datasets: predictions, clear sky and actual

```
In [18]: print('Before cleaning')
         print(df_Actual.shape)
         print(df_Predictions.shape)
         print(df_Clearsky.shape)

         df_Actual=df_Actual[df_Actual['Date'].isin(
             df_Predictions['Date'])]
         df_Actual=df_Actual.reset_index(drop=True)

         df_Predictions=df_Predictions[df_Predictions['Date'].isin(
             df_Actual['Date'])]
         df_Predictions=df_Predictions.reset_index(drop=True)

         df_Clearsky=df_Clearsky[df_Clearsky['Date'].isin(
             df_Predictions['Date'])]
         df_Clearsky=df_Clearsky.reset_index(drop=True)

         print('After cleaning')
         print(df_Actual.shape)
         print(df_Predictions.shape)
         print(df_Clearsky.shape)
```

```
Before cleaning
(12349, 17)
(6698, 12)
(8524, 9)
After cleaning
(6698, 17)
(6698, 12)
(6698, 9)
```

C.3.7.1 Prediction and Actual Correlation Matrix

```
In [19]: PredNS_dataframe=pd.DataFrame()
         PredNS_dataframe['NS Temperature']=df_Actual['Temperature']
         PredNS_dataframe['Pred Temperature']=df_Predictions['Temperature']
         PredNS_dataframe['NS Pressure']=df_Actual['Pressure']
         PredNS_dataframe['Pred Pressure']=df_Predictions['Pressure']
         PredNS_dataframe['NS Humidity']=df_Actual['Humidity']
         PredNS_dataframe['Pred Humidity']=df_Predictions['Humidity']
         PredNS_dataframe['NS Windspeed']=df_Actual['Windspeed']
         PredNS_dataframe['Pred Windspeed']=df_Predictions['Windspeed']

         corrPredNS_dataframe=PredNS_dataframe.corr(
             method='pearson')
         corrPredNS_dataframe.style.background_gradient(
             cmap='coolwarm').set_precision(3)
```

C.3.8 Linear regression dataset

```
In [20]: raw_dataset=pd.DataFrame()

         raw_dataset['Month']=pd.DatetimeIndex(df_Actual['Date']).month
         raw_dataset['Hour']=pd.DatetimeIndex(df_Actual['Date']).hour

         #--------------PREDICTIONS--------------#
         raw_dataset['Temperature']=df_Predictions['Temperature']
         raw_dataset['Windspeed']=df_Predictions['Windspeed']
         raw_dataset['Clouds']=df_Predictions['Clouds']
         raw_dataset['Low Clouds']=df_Predictions['Low Clouds']
         raw_dataset['Medium Clouds']=df_Predictions['Medium Clouds']
         raw_dataset['High Clouds']=df_Predictions['High Clouds']

         #--------------CLEAR SKY--------------#
         raw_dataset['CS Global Insolation']=df_Clearsky['Global Insolation']

         #--------------NORMAL SKY--------------#
         raw_dataset['NS Global Insolation']=df_Actual['Global Insolation']

         raw_dataset.head()

In [21]: corrDS=raw_dataset.drop(['Month', 'Hour'], axis=1).corr(method='pearson')
         corrDS.style.background_gradient(cmap='coolwarm').set_precision(3)
```

C.3.9 Linear Regression to predict solar insolation

```
In [22]: train_dataset = raw_dataset.sample(frac=0.8,random_state=0)
         test_dataset = raw_dataset.drop(train_dataset.index)

In [23]: train_stats=train_dataset.describe()
         train_stats=train_stats.pop('NS Global Insolation')
         train_stats=train_stats.transpose()
         train_stats

Out[23]: count    5358.000000
         mean      220.774023
         std       324.473200
         min         0.000000
         25%         0.000000
         50%         3.891888
         75%       397.557098
         max      1097.628984
         Name: NS Global Insolation, dtype: float64

In [24]: train_labels=train_dataset.pop('NS Global Insolation')
         test_labels=test_dataset.pop('NS Global Insolation')

In [25]: def norm(x):
             return (x - train_stats['mean'])/train_stats['std']
```

```
        normed_train_data=norm(train_dataset)
        normed_test_data=norm(test_dataset)

In [26]: def build_model():
            model = keras.Sequential([
            layers.Dense(32, activation=tf.nn.relu,
                         input_shape=[len(train_dataset.keys())]),
            layers.Dense(16, activation=tf.nn.relu),
            layers.Dense(1)
          ])

            optimizer = tf.keras.optimizers.RMSprop(lr=0.001)

            model.compile(loss='mean_squared_error',
                          optimizer=optimizer,
                          metrics=['mean_absolute_error',
                                   'mean_squared_error'])
            return model

In [27]: model = build_model()

In [28]: model.summary()

In [29]: class PrintDot(keras.callbacks.Callback):
            def on_epoch_end(self, epoch, logs):
                if epoch % 100 == 0:
                    print('')
                print('.', end='')

In [30]: def plot_history(history):
            hist = pd.DataFrame(history.history)
            hist['epoch'] = history.epoch
            plt.figure()
            plt.xlabel('Epoch')
            plt.ylabel('Mean Abs Error [$W/m^2$]')
            plt.plot(hist['epoch'], hist['mean_absolute_error'],
                   label='Train Error')
            plt.plot(hist['epoch'], hist['val_mean_absolute_error'],
                   label = 'Val Error')
            plt.legend()

            plt.figure()
            plt.xlabel('Epoch')
            plt.ylabel('Mean Square Error [$W/m^2$]')
            plt.plot(hist['epoch'], hist['mean_squared_error'],
                   label='Train Error')
            plt.plot(hist['epoch'], hist['val_mean_squared_error'],
                   label = 'Val Error')
            plt.legend()
            plt.show()
```

```
In [31]: EPOCHS = 1000

         # The patience parameter is the amount of epochs to check for improvement
         early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=20)

          history = model.fit(normed_train_data, train_labels,
                              epochs=EPOCHS, validation_split = 0.2,
                              verbose=0, callbacks=[early_stop, PrintDot()])

         plot_history(history)

In [32]: loss, mae, mse = model.evaluate(normed_test_data, test_labels, verbose=0)

         print("Mean Abs Error: {:5.2f}NS Global Insolation".format(mae))

In [33]: test_predictions = model.predict(normed_test_data).flatten()

         plt.scatter(test_labels, test_predictions)
         plt.xlabel('True Values [$W/m^2$]')
         plt.ylabel('Predictions [$W/m^2$]')
         plt.axis('equal')
         plt.axis('square')
         plt.xlim([0,plt.xlim()[1]])
         plt.ylim([0,plt.ylim()[1]])
         _ = plt.plot([-200, 1500], [-200, 1500])

         plt.savefig('TrueVersusPredictions.pdf',bbox_inches = 'tight')

In [34]: error = test_predictions - test_labels
         plt.hist(error, bins = 25)
         plt.xlabel("Prediction Error [$W/m^2$]")
         _ = plt.ylabel("Count")

         plt.savefig('PredictionError.pdf',bbox_inches = 'tight')
```

C.3.10 Linear Regression to predict solar insolation including previous hour information

Note that the original model's data is commented out when running additional information linear regression and the program is restarted entirely to clear the memory.

```
In [35]: raw_dataset['Prev NS'] = df_Actual['Prev NS']

In [36]: train_dataset = raw_dataset.sample(frac=0.8,random_state=0)
         test_dataset = raw_dataset.drop(train_dataset.index)

         train_stats=train_dataset.describe()
         train_stats=train_stats.pop('NS Global Insolation')
         train_stats=train_stats.transpose()
```

```
train_labels=train_dataset.pop('NS Global Insolation')
test_labels=test_dataset.pop('NS Global Insolation')

normed_train_data=norm(train_dataset)
normed_test_data=norm(test_dataset)
```

In [37]:
```
def build_model2():
    model = keras.Sequential([
    layers.Dense(32, activation=tf.nn.relu,
                 input_shape=[len(train_dataset.keys())]),
    layers.Dense(16, activation=tf.nn.relu),
    layers.Dense(1)
  ])

    optimizer = tf.keras.optimizers.RMSprop(lr=0.001)

    model.compile(loss='mean_squared_error',
                  optimizer=optimizer,
                  metrics=['mean_absolute_error',
                           'mean_squared_error'])
    return model


model2 = build_model2()
model2.summary()
```

In [38]:
```
EPOCHS = 1000

early_stop = keras.callbacks.EarlyStopping(monitor='val_loss',
                                           patience=20)

history = model2.fit(normed_train_data, train_labels,
                     epochs=EPOCHS, validation_split = 0.2,
                     verbose=0, callbacks=[early_stop, PrintDot()])

plot_history(history)
```

In [39]:
```
loss, mae, mse = model2.evaluate(normed_test_data, test_labels, verbose=0)

print("Mean Abs Error: {:5.2f}NS Global Insolation".format(mae))
```

In [40]:
```
test_predictions = model2.predict(normed_test_data).flatten()

plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values [$W/m^2$]')
plt.ylabel('Predictions [$W/m^2$]')
plt.axis('equal')
plt.axis('square')
plt.xlim([0,plt.xlim()[1]])
plt.ylim([0,plt.ylim()[1]])
_ = plt.plot([-200, 1500], [-200, 1500])
```

```
        plt.savefig('TrueVersusPredictions_Extra.pdf',bbox_inches = 'tight')
```

```
In [41]: error = test_predictions - test_labels
         plt.hist(error, bins = 25)
         plt.xlabel("Prediction Error [$W/m^2$]")
         _ = plt.ylabel("Count")

         plt.savefig('PredictionError_Extra.pdf',bbox_inches = 'tight')
```

C.4 Reinforcement Learning

C.4.1 Imports

```
In [1]: import warnings
        warnings.filterwarnings('ignore')

        import pandas as pd
        import datetime as dt
        import matplotlib.pyplot as plt
        from time import sleep
        import os
        from yr.libyr import Yr
        import pvlib
        from pvlib.location import Location, solarposition
        import math
        import pytz
        import numpy as np
        import random
        #Variables and constants
        tiltAngle=30
        rho=0.2
        LTM=-30
        latitude=-33.9346
        longitude=18.8668
        altitude=122

        import tensorflow as tf
        from tensorflow import keras
        from tensorflow.keras import layers
        from tensorflow.keras.models import load_model
        # print(tf.__version__)

        loc=Location(latitude,
                     longitude,
                     'Africa/Johannesburg',
                     altitude,
                     'Stellenbosch')

        np.random.seed(30)
```

```
In [2]: label = 20
        ticklabel = 18

        params = {'axes.labelsize': label,
                'axes.titlesize': label,
                'text.usetex'   : True,
                'font.size': ticklabel,
                'legend.frameon': True,
                'legend.fontsize': ticklabel,
                'legend.loc': 'center right',
                'legend.framealpha': 1,
                'xtick.labelsize': ticklabel,
                'ytick.labelsize': ticklabel,
                'grid.linestyle' : ":",
                'axes.grid': True,
                'axes.grid.which': 'major',
                'axes.grid'      : True,
                'grid.linewidth' : 0.8,
                'keymap.grid': ['g'],
                'keymap.grid_minor': ['G'],
                'figure.figsize': [18, 8]
                }
        plt.rcParams.update(params)

In [3]: random.seed(30)
```

C.4.2 Time intervals

```
In [4]: interval = 15
```

C.4.3 Results of GA design

C.4.3.1 SGBG

```
In [5]: # PV
        PV_capacity = 10.6 #kW peak
        efficiency_PV = 16.99/100
        numPV = 29
        PV_area = 1.96 * 0.991

        # Grid
        Grid_hourly = 1.2 #kWh per hour

        # Battery
        DOD = 0.5
        Battery_capacity = 49 # kWh
        fully_charged_kWh = Battery_capacity*DOD # kWh
        initial_charge = fully_charged_kWh
        Max_Discharge_kWh = 0.08 # kWh per minute
        numBat = 15
```

```
        # Generator
        gen_kWh = 5 # kWh per hour

        # Time intervals
        grid_interval = Grid_hourly*(interval/60)
        gen_interval = gen_kWh*(interval/60)
```

C.4.4 Data Preprocessing

```
In [6]: LoadDataPoints=pd.read_table('LoadProfileHousehold.csv',
                                    sep=';' )

        LoadDataPoints=LoadDataPoints.rename(index=str,
                                             columns={'Time':'Date',
                                                      'Sum [kWh]':'Load'
                                                     })
        LoadDataPoints['Date']=pd.to_datetime(LoadDataPoints['Date'])

        LoadDataPoints=LoadDataPoints.drop(['Electricity.Timestep'],
                                           axis=1)


In [7]: LTM=-30
        latitude=-33.9346
        longitude=18.8668
        altitude=122

        #Location
        loc=Location(latitude,
                     longitude,
                     'Africa/Johannesburg',
                     altitude,
                     'Stellenbosch')

        #PV setup
        tiltAngle=30
        rho=0.2

        def getTotalIrradiance(GHI, DNI, DHI, timestamp):
            timestamp=pd.DatetimeIndex(timestamp)
            timestamp=timestamp.tz_localize(loc.tz)
            model='isotropic'
            surface_azimuth=0
            albedo=rho
            latitudeRad=np.deg2rad(latitude)
            n=timestamp.dayofyear
            eqOfTime=pvlib.solarposition.equation_of_time_pvcdrom(n)
            hourangle=pvlib.solarposition.hour_angle(timestamp,
                                                     longitude,
                                                     eqOfTime)
            hourAngleRad=np.deg2rad(hourangle)
            declinationRad=pvlib.solarposition.declination_cooper69(
                n)
```

```
            solar_zenithRad=pvlib.solarposition.solar_zenith_analytical(
                latitudeRad, hourAngleRad, declinationRad)
            solar_azimuthRad=pvlib.solarposition.solar_azimuth_analytical(
                latitudeRad, hourAngleRad, declinationRad,
                solar_zenithRad)
            solar_azimuth = np.rad2deg(solar_azimuthRad)
            solar_zenith= np.rad2deg(solar_zenithRad)
            surface_tilt=tiltAngle
            total=pvlib.irradiance.get_total_irradiance(
                surface_tilt=tiltAngle,
                surface_azimuth=surface_azimuth,
                solar_zenith=solar_zenith,
                solar_azimuth=solar_azimuth,
                dni=DNI,
                ghi=GHI,
                dhi=DHI,
                albedo=albedo,
                model=model)
            return total

        def calcPVPower(insolation):
            kWh=insolation*efficiency_PV*PV_area*numPV/1000
            return kWh/60

In [8]: normalSkyDatapoints = pd.read_table(
            'minutelyWeatherData.csv',
            sep=',' )

        normalSkyDatapoints.head()

        normalSkyDatapoints = normalSkyDatapoints.drop(
            ['Record', 'Date',  'Time',
             'UVA_Avg', 'UVB_Avg', 'Tracker2WM_Avg',
             'WindDir_D1_WVT', 'WindDir_SD1_WVT',
             'ShadowbandWM_Avg', 'RH','AirTC_Avg',
             'WS_ms_S_WVT', 'BP_mB_Avg', 'DNICalc_Avg'],
            axis = 1)

        normalSkyDatapoints=normalSkyDatapoints.rename(
            index = str,
            columns = {'SunWM_Avg':'GHI',
                       'TrackerWM_Avg':'DNI',
                       'ShadowWM_Avg':'DHI',
                       'TimeStamp':'Date'})

        normalSkyDatapoints['Date'] = pd.to_datetime(
            normalSkyDatapoints['Date'])

        normalSkyDatapoints = normalSkyDatapoints.reset_index(
            drop=True)

        start = normalSkyDatapoints.iloc[0]['Date']
```

```
        end = normalSkyDatapoints.iloc[-1]['Date']

        normalSky = pd.DataFrame()

        normalSky['Date'] = pd.date_range(start, end, freq='min')

        normalSkyDF = pd.merge(normalSkyDatapoints, normalSky,
                               on = 'Date', how = 'outer')
        normalSkyDF.shape

        normalSkyDF = normalSkyDF.sort_values(by=['Date'])
        normalSkyDF = normalSkyDF.fillna(method='ffill')

        total = getTotalIrradiance(
            normalSkyDatapoints['GHI'],
            normalSkyDatapoints['DNI'],
            normalSkyDatapoints['DHI'],
            pd.DatetimeIndex(normalSkyDatapoints['Date']))

        normalSkyDF['Global Insolation'] = total['poa_global']

        #kWh for every minute
        normalSkyDF['PV Power'] = calcPVPower(
            normalSkyDF['Global Insolation'])
```

```
In [9]: df=pd.merge(LoadDataPoints,
                    normalSkyDF, on=['Date'])

        df['Date']=pd.to_datetime(df['Date'])

        df=df.drop(
            ['DNI', 'DHI', 'GHI', 'Global Insolation'],
            axis=1)
```

```
In [10]: df = df.set_index('Date')

         df = df.resample('15T').sum()

         df = df.reset_index()
```

```
In [11]: df['HourOfDay'] = df['Date'].dt.hour
         df['Month'] = df['Date'].dt.month

         df['Battery']   = pd.Series(np.zeros(df.shape[0]))
         df['Generator'] = pd.Series(np.zeros(df.shape[0]))

         df['DayofWeek'] = df['Date'].dt.weekday

         df['Weekday'] =df['DayofWeek'].apply(
             lambda x: 1 if x <= 4 else 0)
```

```
In [12]: df['Month'] = pd.Categorical(df['Month'])
         df['Month'] = df.Month.cat.codes

         df['HourOfDay'] = pd.Categorical(df['HourOfDay'])
         df['HourOfDay'] = df.HourOfDay.cat.codes

         df['Weekday'] = pd.Categorical(df['Weekday'])
         df['Weekday'] = df.Weekday.cat.codes

In [13]: df = df.drop(['DayofWeek'], axis=1)

In [14]: df.tail()

In [15]: for i in range(0, len(df)-1):
             df.loc[i, 'Next PV'] = df.loc[i+1, 'PV Power']
             df.loc[i, 'Next Load'] = df.loc[i+1, 'Load']
```

C.4.5 Train, validate and test sets

```
In [16]: df_train_set = pd.DataFrame()
         df_valid_set = pd.DataFrame()
         df_test_set = pd.DataFrame()

         for j in range (1,13):
             year = 2018
             month = j
             train_day_start = 1
             train_day_end = 22
             valid_day_start = 23
             valid_day_end = 26
             test_start = 27
             test_end = 0

             days_31 = [1, 3, 5, 7, 8, 10, 12]
             days_30 = [4, 6, 9, 11]

             # February
             if j == 2:                    # watch out for February
                 test_end = 28
             # 31: January, March, May, July, Aug, Oct, Dec
             if j in days_31:
                 test_end = 31
             if j in days_30:
                 test_end = 30

              # TRAIN DATA SEGMENT
             start_date = pd.Timestamp(year=year, month=month,
                                       day=train_day_start,
                                       hour = 0, minute = 0,
                                       second = 0)
             end_date = pd.Timestamp(year=year, month=month,
```

```
                                    day=train_day_end, hour = 23,
                                    minute = 59, second = 59)

            dti = pd.date_range(start = start_date,
                                end = end_date, freq='15T')
            df_train_set = df_train_set.append(
                df.loc[df.Date.isin(pd.DatetimeIndex(dti))])

            # VALID DATA SEGMENT
            start_date = pd.Timestamp(year=year, month=month,
                                      day=valid_day_start,
                                      hour = 0, minute = 0,
                                      second = 0)
            end_date = pd.Timestamp(year=year, month=month,
                                    day=valid_day_end, hour = 23,
                                    minute = 59, second = 59)

            dti = pd.date_range(start = start_date,
                                end = end_date, freq='15T')
            df_valid_set = df_valid_set.append(
                df.loc[df.Date.isin(pd.DatetimeIndex(dti))])

            # TEST DATA SEGMENT
            start_date = pd.Timestamp(year=year, month=month,
                                      day=test_start, hour = 0,
                                      minute = 0, second = 0)
            end_date = pd.Timestamp(year=year, month=month,
                                    day=test_end, hour = 23,
                                    minute = 59, second = 59)

            dti = pd.date_range(start = start_date,
                                end = end_date, freq='15T')
            df_test_set = df_test_set.append(
                df.loc[df.Date.isin(pd.DatetimeIndex(dti))])
```

```
In [17]: df_train_set = df_train_set.drop(['Date'], axis=1)
         df_valid_set = df_valid_set.drop(['Date'], axis=1)
         df_test_set = df_test_set.drop(['Date'], axis=1)
```

```
In [18]: df_train_set.to_csv('RL Train.csv')
         df_valid_set.to_csv('RL Valid.csv')
         df_test_set.to_csv('RL Test.csv')
```

```
In [19]: # df_train_set = pd.read_csv('RL Train.csv')
         # df_train_set = df_train_set.drop(['Unnamed: 0'], axis=1)

         # df_valid_set = pd.read_csv('RL Valid.csv')
         # df_valid_set = df_valid_set.drop(['Unnamed: 0'], axis=1)

         # df_test_set = pd.read_csv('RL Test.csv')
         # df_test_set = df_test_set.drop(['Unnamed: 0'], axis=1)
```

In [20]:
```
# print('Train:\t\t{0}\t{1:.1f}%'.format(
#     df_train_set.shape[0],
#     100*df_train_set.shape[0]/df.shape[0]))
# print('Validate:\t{0}\t{1:.1f}%'.format(
#     df_valid_set.shape[0],
#     100*df_valid_set.shape[0]/df.shape[0]))
# print('Test:\t\t{0}\t{1:.1f}%'.format(
#     df_test_set.shape[0],
#     100*df_test_set.shape[0]/df.shape[0]))
```

C.4.6 Actions and state space

In [21]:
```
action_perm  = {'use_pv': {True, False},
                'charge_bat': {True, False},
                'discharge_bat': {True, False},
                'use_grid': {True, False},
                'switch_on_Gen': {True, False}}

# remove actions charge + discharge == True
import itertools
l = ['T', 'F']
actions_all = [list(i) for i in itertools.product(l, repeat=5)]

remove_ = 8

te = 0

while True:
    for i in range (0, len(actions_all)):
        if actions_all[i][1] == 'T' and actions_all[i][2] == 'T':
            actions_all.remove(actions_all[i])
            te += 1
            break
    if te == remove_:
        break

actions = []
for i in range (0, len(actions_all)):
    actions.append(actions_all[i][0] + actions_all[i][1] \
                   + actions_all[i][2]+ actions_all[i][3] \
                   + actions_all[i][4])
```

In [22]:
```
#define episode
environment  = {'pv_power': None,
                'load': None,
                'month': None,
                'hour_of_day': None,
                'battery': None,
                'generator': None,
                'weekday': None}
```

In [23]:
```
train_set = df_train_set.as_matrix()
```

```
valid_set = df_valid_set.as_matrix()
test_set = df_test_set.as_matrix()
```

C.4.7 Controller variables

```
In [24]: num_actions=len(actions)
```

```
In [25]: num_states = 7
```

```
In [26]: def step(action, t, prev_state, data):

             next_state    = None

             action        = list(action)
             use_pv        = action[0]
             charge_bat    = action[1]
             discharge_bat = action[2]
             use_grid      = action[3]
             use_gen       = action[4]

             # Load and PV for the time step
             PV    = data[t][7]
             load  = data[t][8]

             # Where the battery and generator are starting
             bat   = prev_state[4]
             gen   = prev_state[5]

             bat_before = bat

             grid  = grid_interval # kWh
             LPS   = load #100%
             maxGen = gen_interval #kWh

             PV_Use  = 0
             Gen_Use = 0
             Grid_Use = 0
             Bat_Use = 0
             availablePV = 0
             availableGrid = 0
             availableGen = 0

             ##############################
             # Use PV Action
             if use_pv == 'T':
                 availablePV = PV
                 if availablePV > load:
                     PV_Use += LPS
                     availablePV -= LPS
                     LPS = 0
                 else:
```

```
            LPS -= availablePV
            PV_Use += availablePV
            availablePV = 0
    ############################
    # Discharge Battery Action
    if discharge_bat == 'T':
        if LPS > 0:
            if LPS < bat:
                bat -= LPS
                Bat_Use += LPS
                LPS = 0
            else:
                LPS -=bat
                Bat_Use += bat
                bat = 0
    ############################
    # Use Grid Action
    if use_grid == 'T':
        availableGrid = grid
        if LPS > 0:
            if LPS < availableGrid:
                availableGrid -= LPS
                Grid_Use += LPS
                LPS = 0
            else:
                LPS -= availableGrid
                Grid_Use += availableGrid
                availableGrid = 0
    ############################
    # Use Generator Action
    if use_gen == 'T':
        availableGen = maxGen
        if gen < (15/interval):
            availableGen = 0.2*availableGen
        if LPS > 0:
            if LPS < availableGen:
                availableGen -= LPS
                Gen_Use += LPS
                LPS = 0
            else:
                LPS -= availableGen
                Grid_Use += availableGen
                availableGen = 0
    ############################
    # Charge batteries Action
    #maxC = MaxCharge_kWh
    if charge_bat == 'T':
        if use_pv == 'T':
            if bat+availablePV > fully_charged_kWh:
                PV_Use += fully_charged_kWh - bat
                bat = fully_charged_kWh
            else:
```

```
                bat += availablePV
                PV_Use += availablePV
        if use_grid == 'T':
            if bat+availableGrid > fully_charged_kWh:
                Grid_Use += fully_charged_kWh - bat
                bat = fully_charged_kWh
            else:
                bat += availableGrid
                Grid_Use += availableGrid
        if use_gen == 'T':
            if bat+availableGen > fully_charged_kWh:
                Gen_Use += fully_charged_kWh - bat
                bat = fully_charged_kWh
            else:
                bat += availableGen
                PV_Use += availableGen
    ############################
    battery_charged = bat - bat_before
    battery_charged = max(0, battery_charged)

    # Rewards
    reward = 0
    reward = reward + int(LPS <= 0)*20
    reward = reward + int(PV_Use == PV and use_pv == 'T')*6
    reward = reward + int(PV_Use > 0)*6
    reward = reward + int(use_gen == 'T' and PV == 0)*3
    reward = reward + int(Bat_Use > 0 and PV == 0)*6

    if use_gen == 'F':
        gen = 0 # switch off generator
    else:
        gen += 1

    used_energy = PV_Use + Grid_Use + Gen_Use + Bat_Use
    net_energy = load - used_energy - LPS + battery_charged
    LPS = max(0, LPS)

    # THE NEXT STATE
    next_state    = [data[t][8],
                     data[t][7],
                     data[t][2],
                     data[t][3],
                     bat,
                     gen,
                     data[t][6]]

    if t+1 == len(data):
        next_state = None

    return next_state, reward, t+2 == len(
        data), LPS, PV_Use, Gen_Use, Grid_Use, Bat_Use, battery_charged, None
```

```
In [27]: # number of episodes
         num_episodes = 50
```

C.4.8 Training

```
In [28]: memory = []

         max_memory = int(2*len(train_set))

         def sampleMemory(batch_size):
             if batch_size > len(memory):
                 return random.sample(memory,
                                      (len(memory)))
             else:
                 return random.sample(memory,
                                      batch_size)

         def addSampleMemory(memorySample):
             memory.append(memorySample)
             if len(memory) > max_memory:
                 memory.pop(0)
```

```
In [29]: max_epsilon = 1
         min_epsilon = 0.01

         Lambda = 0.001
         gamma  = 0.7

         batch_hours = 3
         batch_size  = int((60/interval)*batch_hours)
```

```
In [30]: def build_model():
             model = keras.Sequential(
             [layers.Dense(32, activation=tf.nn.relu,
                           input_shape=(num_states,)),
              layers.Dense(32, activation=tf.nn.relu),
              layers.Dense(32, activation=tf.nn.relu),
              layers.Dense(num_actions)
             ])

             return model
```

```
In [31]: model = build_model()

         model.compile(loss='mse',
                       optimizer=tf.keras.optimizers.Adam(
                           lr=Lambda))

         print(model.summary())
```

```
In [32]: total_LPS = np.zeros(num_episodes)
         total_rewards = np.zeros(num_episodes)
         total_PV = np.zeros(num_episodes)
         total_Gen = np.zeros(num_episodes)
         total_Grid = np.zeros(num_episodes)
         total_BatU = np.zeros(num_episodes)
         total_BatC = np.zeros(num_episodes)
```

```
In [33]: def RLController_Train():
             epsilon  = max_epsilon
             cnt      = 0
             steps    = 0

             data = train_set

             while cnt < num_episodes:
                 print('Episode {} of {}'.format(cnt, num_episodes))

                 # reset state to start of dataset
                 state    = [data[0][0],
                             data[0][1],
                             data[0][2],
                             data[0][3],
                             data[0][4],
                             data[0][5],
                             data[0][6]]
                 state[4] = initial_charge
                 state[5] = 0 #off

                 # save episode
                 episode_rewards = 0
                 episode_LPS = 0
                 episode_PV = 0
                 episode_Gen = 0
                 episode_Grid = 0
                 episode_BatU = 0
                 episode_BatC = 0

                 # run entire episode
                 for t in range(len(data)):
                     action=0

                     if random.random() < epsilon:
                         action = random.randint(0, num_actions - 1)
                     else:
                         options = model.predict(np.array([state]))
                         action = np.argmax(options)

                     # step through data
                     next_state, reward, done, LPS, PV, Gen, Grid, BatU, BatC, _ = step(
                         actions[action], t, state, data)
```

```
addSampleMemory(
    [state, action, reward, next_state])

random_batch = sampleMemory(batch_size)

states_ = np.array(
        [val[0] for val in random_batch])

next_states = np.array(
        [(np.zeros(num_states)
          if val[3] is None else val[3]) for val in random_batch])

q_s_a = model.predict(states_)

q_s_a_d = model.predict(next_states)

x_batch = np.zeros((len(random_batch),
                        num_states))

y_batch = np.zeros((len(random_batch),
                        num_actions))

for i, b in enumerate(random_batch):
    state_B, action_B, reward_B, next_state_B = b[0], b[1], b[2], b[3]

    current_q_B = q_s_a[i]

    if next_state == None:
        current_q_B[action_B] = reward_B
    else:
        current_q_B[action_B]=reward_B + gamma*np.amax(
            q_s_a_d[i])

    x_batch[i] = state_B
    y_batch[i] = current_q_B

model.fit(x=x_batch, y=y_batch, verbose=0)

steps +=1

state = next_state

epsilon = min_epsilon + (max_epsilon-min_epsilon)*math.exp(
    -Lambda*steps)

episode_rewards += reward
episode_LPS += LPS
episode_PV += PV
episode_Gen += Gen
episode_Grid += Grid
episode_BatU += BatU
episode_BatC += BatC
```

```
                    if done:
                        total_rewards[cnt]=episode_rewards
                        total_LPS[cnt]=episode_LPS
                        total_PV[cnt]=episode_PV
                        total_Gen[cnt]=episode_Gen
                        total_Grid[cnt] = episode_Grid
                        total_BatU[cnt] = episode_BatU
                        total_BatC[cnt] = episode_BatC
                        break

                cnt += 1
```

```
In [34]: RLController_Train()
```

```
In [35]: model.save('model.h5')
```

```
In [36]: plt.plot(total_rewards)
         plt.xlabel('Episodes')
         plt.ylabel('Total rewards obtained per episode during training')
         plt.savefig('Training_rewards.pdf',bbox_inches = 'tight')
         #plt.show()
```

```
In [37]: plt.plot(total_LPS)
         plt.xlabel('Episodes')
         plt.ylabel('Total LPS [kWh] per episode during training')
         plt.savefig('Training_LPS.pdf',bbox_inches = 'tight')
         #plt.show()
```

```
In [38]: plt.plot(total_PV)
         plt.plot(total_BatU)
         plt.plot(total_Gen)
         plt.plot(total_Grid)
         plt.xlabel('Episodes')
         plt.ylabel('Total Power [kWh] per episode during training')
         plt.legend(['PV', 'Battery', 'Generator', 'Grid'])
         plt.savefig('Training_HPS.pdf',bbox_inches = 'tight')
         #plt.show()
```

C.4.9 Validation

```
In [39]: validate_Baseline1  = []
         validate_Baseline2  = []
         validate_RL = []
```

C.4.9.1 Baseline controller 1

```
In [40]: def baselineController1(data, Process):
             cnt = 0
             steps = 0
```

```
# reset state to start of dataset
state    = [data[0][0],
            data[0][1],
            data[0][2],
            data[0][3],
            data[0][4],
            data[0][5],
            data[0][6]]
state[4] = initial_charge
state[5] = 0 # off

episode_rewards = 0
episode_LPS = 0
episode_PV = 0
episode_Gen = 0
episode_Grid = 0
episode_BatU = 0
episode_BatC = 0

# run entire episode
for t in range(0,len(data)):
    # random action
    action = random.randint(0, num_actions - 1)

    # step through data
    next_state, reward, done, LPS, PV, Gen, Grid, BatU, BatC, _ = step(
        actions[action], t, state, data)

    steps +=1

    state = next_state

    episode_LPS += LPS
    episode_rewards += reward
    episode_PV += PV
    episode_Gen += Gen
    episode_Grid += Grid
    episode_BatU += BatU
    episode_BatC += BatC

    if done:
        if Process == 'Validate':
            global validate_Baseline1
            validate_Baseline1 = [episode_rewards,
                                  episode_LPS,
                                  episode_BatU,
                                  episode_PV,
                                  episode_Gen,
                                  episode_Grid,
                                  episode_BatC]
```

```
                    if Process == 'Test':
                        global test_Baseline1
                        test_Baseline1 = [episode_rewards,
                                          episode_LPS,
                                          episode_BatU,
                                          episode_PV,
                                          episode_Gen,
                                          episode_Grid,
                                          episode_BatC]
                    break
```

C.4.9.2 Baseline controller 2

Rule-based controller

```
In [41]: def baselineController2(data, Process):
             cnt = 0
             steps = 0

             # reset state to start of dataset
             state    = [data[0][0],
                         data[0][1],
                         data[0][2],
                         data[0][3],
                         data[0][4],
                         data[0][5],
                         data[0][6]]

             state[4] = initial_charge
             state[5] = 0 #off

             episode_rewards = 0
             episode_LPS = 0
             episode_PV = 0
             episode_Gen = 0
             episode_Grid = 0
             episode_BatU = 0
             episode_BatC = 0

             for t in range(len(data)):

                 # rule-based action
                 use_pv        = None
                 charge_bat    = None
                 discharge_bat = None
                 use_grid      = None
                 use_gen       = None

                 # use PV
                 if state[1] > 0:
                     use_pv = 'T'
                 else:
```

```
        use_pv = 'F'
    # charge bat if PV > (load)
    if state[1] > state[0]:
        charge_bat = 'T'
    else:
        charge_bat = 'F'
    # discharge bat if PV < load
    if state[1] < state[0] and charge_bat != 'T':
        discharge_bat = 'T'
    else:
        discharge_bat = 'F'
    # use grid randomly
    if bool(random.getrandbits(1)) == 1:
        use_grid = 'T'
    else:
        use_grid = 'F'
    # use generator randomy
    if bool(random.getrandbits(1)) == 1:
        use_gen = 'T'
    else:
        use_gen = 'F'

    action = use_pv+charge_bat+discharge_bat+use_grid+use_gen

    # step through data
    next_state, reward, done, LPS, PV, Gen, Grid, BatU, BatC,_ = step(
        action, t, state, data)

    steps += 1

    state = next_state

    episode_rewards += reward
    episode_LPS += LPS
    episode_PV += PV
    episode_Gen += Gen
    episode_Grid += Grid
    episode_BatU += BatU
    episode_BatC += BatC

    if done:
        if Process == 'Validate':
            global validate_Baseline2
            validate_Baseline2 = [episode_rewards,
                                  episode_LPS,
                                  episode_BatU,
                                  episode_PV,
                                  episode_Gen,
                                  episode_Grid,
                                  episode_BatC]

        if Process == 'Test':
```

```
                    global test_Baseline2
                    test_Baseline2 = [episode_rewards,
                                      episode_LPS,
                                      episode_BatU,
                                      episode_PV,
                                      episode_Gen,
                                      episode_Grid,
                                      episode_BatC]
                break
```

C.4.10 RL Controller

```
In [42]: def RLController(data, Process):
             epsilon  = max_epsilon
             cnt      = 0
             steps    = 0

             # reset state to start of dataset
             state    = [data[0][0],
                         data[0][1],
                         data[0][2],
                         data[0][3],
                         data[0][4],
                         data[0][5],
                         data[0][6]]
             state[4] = initial_charge
             state[5] = 0 #off

             # save episode
             episode_rewards = 0
             episode_LPS = 0
             episode_PV = 0
             episode_Gen = 0
             episode_Grid = 0
             episode_BatU = 0
             episode_BatC = 0

             if Process == 'Test':
                 global model
                 del model
                 model = keras.models.load_model('model.h5')

             # run entire episode
             for t in range(len(data)):
                 action=0

                 if random.random() < epsilon:
                     action = random.randint(0, num_actions - 1)
                 else:
                     options = model.predict(np.array([state]))
                     action = np.argmax(options)
```

```
# step through data
next_state, reward, done, LPS, PV, Gen, Grid, BatU, BatC, _ = step(
    actions[action], t, state, data)

addSampleMemory(
    [state, action, reward, next_state])

random_batch = sampleMemory(batch_size)

states_ = np.array(
        [val[0] for val in random_batch])

next_states = np.array(
        [(np.zeros(num_states)
          if val[3] is None else val[3]) for val in random_batch])

q_s_a = model.predict(states_)

q_s_a_d = model.predict(next_states)

x_batch = np.zeros((len(random_batch),
                        num_states))

y_batch = np.zeros((len(random_batch),
                        num_actions))

for i, b in enumerate(random_batch):
    state_B, action_B, reward_B, next_state_B = b[0], b[1], b[2], b[3]

    current_q_B = q_s_a[i]

    if next_state == None:
        current_q_B[action_B] = reward_B
    else:
        current_q_B[action_B]=reward_B + gamma*np.amax(
            q_s_a_d[i])

    x_batch[i] = state_B
    y_batch[i] = current_q_B

model.fit(x=x_batch, y=y_batch, verbose=0)

steps +=1

state = next_state

epsilon = min_epsilon + (max_epsilon-min_epsilon)*math.exp(
    -Lambda*steps)

episode_rewards += reward
episode_LPS += LPS
episode_PV += PV
```

```
                episode_Gen += Gen
                episode_Grid += Grid
                episode_BatU += BatU
                episode_BatC += BatC

                if done:
                    if Process == 'Validate':
                        global validate_RL
                        validate_RL = [episode_rewards,
                                       episode_LPS,
                                       episode_BatU,
                                       episode_PV,
                                       episode_Gen,
                                       episode_Grid,
                                       episode_BatC]

                    if Process == 'Test':
                        global test_RL
                        test_RL = [episode_rewards,
                                   episode_LPS,
                                   episode_BatU,
                                   episode_PV,
                                   episode_Gen,
                                   episode_Grid,
                                   episode_BatC]

                cnt += 1
```

```
In [43]: baselineController1(valid_set, 'Validate')
         baselineController2(valid_set, 'Validate')
```

```
In [44]: RLController(valid_set, 'Validate')
```

```
In [45]: validationResults = pd.DataFrame(columns=['Simulation',
                                                   'Rewards',
                                                   'LPS',
                                                   'BatU',
                                                   'PV',
                                                   'Gen',
                                                   'Grid',
                                                   'BatC'])

         totalLoad = df_valid_set['Load'].sum()

         B1_validationResults = {'Simulation': 'Baseline1',
                                 'Rewards': validate_Baseline1[0],
                                 'LPS': validate_Baseline1[1],
                                 'BatU': validate_Baseline1[2],
                                 'PV': validate_Baseline1[3],
                                 'Gen': validate_Baseline1[4],
                                 'Grid': validate_Baseline1[5],
                                 'BatC': validate_Baseline1[6]}
```

```
B2_validationResults = {'Simulation': 'Baseline2',
                        'Rewards': validate_Baseline2[0],
                        'LPS': validate_Baseline2[1],
                        'BatU': validate_Baseline2[2],
                        'PV': validate_Baseline2[3],
                        'Gen': validate_Baseline2[4],
                        'Grid': validate_Baseline2[5],
                        'BatC': validate_Baseline2[6]}

RL_validationResults = {'Simulation': 'RL',
                        'Rewards': validate_RL[0],
                        'LPS': validate_RL[1],
                        'BatU': validate_RL[2],
                        'PV': validate_RL[3],
                        'Gen': validate_RL[4],
                        'Grid': validate_RL[5],
                        'BatC': validate_RL[6]}

validationResults = validationResults.append(
    B1_validationResults, ignore_index=True)
validationResults = validationResults.append(
    B2_validationResults, ignore_index=True)
validationResults = validationResults.append(
    RL_validationResults, ignore_index=True)

print('Total Load {}'.format(df_valid_set['Load'].sum()))

usedB1 = validate_Baseline1[2] + validate_Baseline1[3]+ \
            validate_Baseline1[4] + validate_Baseline1[5]
chargedB1 = validate_Baseline1[6]
netB1 = totalLoad - usedB1 - validate_Baseline1[1] + chargedB1
print('Net B1 {} '.format(netB1))

usedB2 = validate_Baseline2[2] + validate_Baseline2[3] + \
        validate_Baseline2[4] + validate_Baseline2[5]
chargedB2 = validate_Baseline2[6]
netB2 = totalLoad - usedB2 - validate_Baseline2[1] + chargedB2
print('Net B2 {} '.format(netB2))

usedRL = validate_RL[2] + validate_RL[3] + validate_RL[4] \
    + validate_RL[5]
chargedRL = validate_RL[6]
netRL = totalLoad - usedRL - validate_RL[1] + chargedRL
print('Net RL {} '.format(netRL))
```

```
Total Load 1191.7470360191815
```

```
In [46]: validationResults
```

```
Out[46]:   Simulation Rewards         LPS        BatU           PV         Gen  \
         0  Baseline1   100231  240.037551  300.262726   489.873112  176.776575
```

```
1  Baseline2  126975   13.124372  524.675107  1088.479151   10.144639
2         RL  134807   58.319839  477.939337   906.990113   38.422668

          Grid        BatC
0  284.890595  300.262726
1   77.176853  522.022289
2  184.344225  474.438349
```

C.4.11 Test

```
In [47]: test_Baseline1 = []
         test_Baseline2 = []
         test_RL = []
```

```
In [48]: baselineController1(test_set, 'Test')
         baselineController2(test_set, 'Test')
         RLController(test_set, 'Test')
```

```
In [49]: testResults = pd.DataFrame(columns=['Simulation',
                                             'Rewards',
                                             'LPS',
                                             'BatU',
                                             'PV',
                                             'Gen',
                                             'Grid',
                                             'BatC'])

         totalLoad = df_test_set['Load'].sum()

         B1_testResults = {'Simulation': 'Baseline1',
                           'Rewards': test_Baseline1[0],
                           'LPS': test_Baseline1[1],
                           'BatU': test_Baseline1[2],
                           'PV': test_Baseline1[3],
                           'Gen': test_Baseline1[4],
                           'Grid': test_Baseline1[5],
                           'BatC': test_Baseline1[6]}

         B2_testResults = {'Simulation': 'Baseline2',
                           'Rewards': test_Baseline2[0],
                           'LPS': test_Baseline2[1],
                           'BatU': test_Baseline2[2],
                           'PV': test_Baseline2[3],
                           'Gen': test_Baseline2[4],
                           'Grid': test_Baseline2[5],
                           'BatC': test_Baseline2[6]}

         RL_testResults = {'Simulation': 'RL',
                           'Rewards': test_RL[0],
                           'LPS': test_RL[1],
                           'BatU': test_RL[2],
```

```
                        'PV': test_RL[3],
                        'Gen': test_RL[4],
                        'Grid': test_RL[5],
                        'BatC': test_RL[6]}

        testResults = testResults.append(
            B1_testResults, ignore_index=True)
        testResults = testResults.append(
            B2_testResults, ignore_index=True)
        testResults = testResults.append(
            RL_testResults, ignore_index=True)

        print('Total Load {}'.format(df_valid_set['Load'].sum()))

        usedB1 = test_Baseline1[2] + test_Baseline1[3] \
                + test_Baseline1[4] + test_Baseline1[5]
        chargedB1 = test_Baseline1[6]
        netB1 = totalLoad - usedB1 - test_Baseline1[1] + chargedB1
        print('Net B1 {} '.format(netB1))

        usedB2 = test_Baseline2[2] + test_Baseline2[3] \
                + test_Baseline2[4] + test_Baseline2[5]
        chargedB2 = test_Baseline2[6]
        netB2 = totalLoad - usedB2 - test_Baseline2[1] + chargedB2
        print('Net B2 {} '.format(netB2))

        usedRL = test_RL[2] + test_RL[3] + test_RL[4] + test_RL[5]
        chargedRL = test_RL[6]
        netRL = totalLoad - usedRL - test_RL[1] + chargedRL
        print('Net RL {} '.format(netRL))
```

```
Total Load 1191.7470360191815
```

```
In [50]: testResults
```

```
Out[50]:   Simulation Rewards         LPS        BatU           PV         Gen  \
        0  Baseline1  109158  253.882847  305.803353   539.701649  190.251698
        1  Baseline2  140337   12.056318  577.311400  1203.450427   10.664405
        2         RL  150130   54.623707  523.951993  1001.605694   54.521456

                Grid        BatC
        0  330.450405  305.246048
        1   82.476853  571.115499
        2  197.841246  517.700190
```

C.5 Reinforcement Learning with IoT-implementation

C.5.1 Imports

In [1]:
```
import time
import warnings
warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow.keras.backend as K
K.clear_session()

import pvlib
from pvlib.location import Location, solarposition

import datetime as dt

import numpy as np
import math
import pandas as pd
import os
import csv
import sys
import random

import glob

import itertools
import matplotlib

from collections import defaultdict

tiltAngle=30
rho=0.2
LTM=-30
latitude=-33.9346
longitude=18.8668
altitude=122

loc=Location(latitude,
             longitude,
             'Africa/Johannesburg',
             altitude,
             'Stellenbosch')

random.seed(30)
```

```
In [2]: label = 20
        ticklabel = 18

        params = {'axes.labelsize': label,
                'axes.titlesize': label,
                'text.usetex'   : True,
                'font.size': ticklabel,
                'legend.frameon': True,
                'legend.fontsize': ticklabel,
                'legend.loc': 'center right',
                'legend.framealpha': 1,
                'xtick.labelsize': ticklabel,
                'ytick.labelsize': ticklabel,
                'grid.linestyle' : ":",
                'axes.grid': True,
                'axes.grid.which': 'major',
                'axes.grid'      : True,
                'grid.linewidth' : 0.8,
                'keymap.grid': ['g'],
                'keymap.grid_minor': ['G'],
                'figure.figsize': [18, 8]
                }
        plt.rcParams.update(params)
```

C.5.2 Results of GA design

C.5.2.1 SGBG

```
In [3]: interval = 15
```

```
In [4]: # PV
        PV_capacity = 10.6 #kW peak
        efficiency_PV = 16.99/100
        numPV = 29
        PV_area = 1.96 * 0.991

        # Grid
        Grid_hourly = 1.2 #kWh per hour

        # Battery
        DOD = 0.5
        Battery_capacity = 49 # kWh
        fully_charged_kWh = Battery_capacity*DOD # kWh
        initial_charge = fully_charged_kWh
        Max_Discharge_kWh = 0.08 # kWh per minute
        numBat = 15

        # Generator
        gen_kWh = 5.0 # kWh per hour

        # Time intervals
        grid_interval = Grid_hourly*(interval/60)
```

```
Max_Discharge_kWh = Max_Discharge_kWh*interval
MaxCharge_kWh = 12*10*numBat*(interval/60)/1000
gen_interval = gen_kWh*(interval/60)

def calcPVPower(insolation):
    kWh=insolation*efficiency_PV*PV_area*numPV/1000
    return kWh/60

def normaliseInterval(unnormalised):
    return unnormalised/interval
```

C.5.3 Data preprocessing

```
In [5]: def getTotalIrradiance(GHI, DNI, DHI, timestamp):
            timestamp=pd.DatetimeIndex(timestamp)
            timestamp=timestamp.tz_localize(loc.tz)
            """
            Calculate the total irradiance

            Args:
                GHI: global horizontal irradiance
                DNI: direct normal irradiance
                DHI: diffuse horizontal irradiance
                Timestamp: DatetimeIndex

            Returns:
                total: [poa_global,
                poa_direct,
                poa_diffuse,
                poa_sky_diffuse,
                poa_ground_diffuse]
            """

            model='isotropic'

            # Panel azimuth from North
            surface_azimuth=0

            albedo=rho #surface albedo - declared in variables and constants

            # convert to radians
            latitudeRad=np.deg2rad(latitude)
            n=timestamp.dayofyear
            # Equation of time
            """
            pvlib.solarposition.equation_of_time_pvcdrom
                Parameters: dayofyear
                Return: equation_of_time: in minutes
            """
            eqOfTime=pvlib.solarposition.equation_of_time_pvcdrom(n)

            # Hour angle
```

```
"""
pvlib.solarposition.hour_angle
    Parameters: times: pd.DatetimeIndex must be
    localized to timezone for the longitude
                longitude: longitude (degrees)
                equation_of_time: equation of time in minutes
    Returns: hour_angle (degrees)
"""
hourangle=pvlib.solarposition.hour_angle(timestamp,
                                         longitude,
                                         eqOfTime)
hourAngleRad=np.deg2rad(hourangle)

#Declination angle calculations
"""
pvlib.solarposition.declination_cooper69
    Parameters: dayofyear
    Returns: declination angle (radians)
"""
declinationRad=pvlib.solarposition.declination_cooper69(n)

#Solar zenith angle calculations
"""
pvlib.solarposition.solar_zenith_analytical
    Parameters: latitude: Latitude of location (radians)
                hourangle: Hour angle in the local solar time (radians)
                declination: declination of the sun (radians)
    Returns: solar zenith angle (radians)
"""
solar_zenithRad=pvlib.solarposition.solar_zenith_analytical(latitudeRad,
                                                            hourAngleRad,
                                                            declinationRad)

#Solar azimuth angle calculations
"""
pvlib.solarposition.solar_azimuth_analytical
    Parameters: latitude: Latitude of location (radians)
                hourangle: Hour angle in the local solar time (radians)
                declination: declination of the sun (radians)
                zenith: solar zenith angle (radians)
    Returns: Solar azimuth angle (radians)

 """
solar_azimuthRad=pvlib.solarposition.solar_azimuth_analytical(latitudeRad,
                                                              hourAngleRad,
                                                              declinationRad,
                                                              solar_zenithRad)
#convert to degrees
solar_azimuth = np.rad2deg(solar_azimuthRad)
solar_zenith= np.rad2deg(solar_zenithRad)
surface_tilt=tiltAngle
# get total irradiance
```

```
            """
            pvlib.irradiance.get_total_irradiance
                Parameters: surface_tilt: Panel tilt from horizontal (degrees)
                            surface_azimuth: Panel azimuth from north (degrees)
                            solar_zenith: Solar zenith angle (degrees)
                            dni: Direct Normal Irradiance (W/m^2)
                            ghi:Global horizontal irradiance (W/m^2)
                            dhi: Diffuse horizontal irradiance (W/m^2)
                            dni_extra: Extraterrestrial direct normal irradiance.
                            airmass: Airmass
                            albedo: Surface albedo 0-1
                            surface_type: Surface type.
                            model: isotropic(default), klucher, haydavies, reindl, king, perez
                Returns: dataframe array:
                            poa_global,
                            poa_direct,
                            poa_diffuse,
                            poa_sky_diffuse,
                            poa_ground_diffuse
            """
            total=pvlib.irradiance.get_total_irradiance(surface_tilt=tiltAngle,
                                                        surface_azimuth=surface_azimuth,
                                                        solar_zenith=solar_zenith,
                                                        solar_azimuth=solar_azimuth,
                                                        dni=DNI,
                                                        ghi=GHI,
                                                        dhi=DHI,
                                                        albedo=albedo,
                                                        model=model)

            return total
```

```
In [6]: path = 'C:/Users/18296033/Documents/WORK/DataCollector/datapoints/Batch2'
        all_files = glob.glob(path + "/*.csv")

        df = pd.DataFrame()

        for filename in all_files:
            temp = pd.read_csv(filename, index_col=None, header=0)
            df=df.append(temp)

        df['From'] = pd.to_datetime(df['From'])
        df['Timestamp'] = pd.to_datetime(df['Timestamp'])

        df = df.sort_values(by=['From','Timestamp'], ascending=[True, True])

        df_Predictions=df[df['From'] == df['Timestamp']]
        df_Predictions = df_Predictions.rename(index=str, columns = {'From':'Date'})
        df_Predictions = df_Predictions.reset_index()
        df_Predictions = df_Predictions.drop(['Timestamp', 'index'], axis=1)
```

```
In [7]: start=df_Predictions.iloc[0]['Date']
```

```
        end=df_Predictions.iloc[-1]['Date']

        timesClearSky=pd.date_range(start, end, freq='T',
                          tz=loc.tz)

        df_Clearsky=loc.get_clearsky(timesClearSky)
        df_Clearsky['Date'] = df_Clearsky.index.get_values()
        df_Clearsky['Date'] = pd.to_datetime(df_Clearsky['Date']) + pd.to_timedelta(2,unit='h')

        df_Clearsky=df_Clearsky.reset_index(drop=True)

        total=getTotalIrradiance(df_Clearsky['ghi'],
                                 df_Clearsky['dni'],
                                 df_Clearsky['dhi'],
                                 pd.DatetimeIndex(df_Clearsky['Date']))

        df_Clearsky['Insolation_CS'] = total['poa_global']

        df_Clearsky = df_Clearsky.drop(['ghi', 'dni', 'dhi'], axis=1)

        temp = pd.date_range(start, end, freq='1h',
                       tz=loc.tz)

        temp = temp.tz_localize(None)

        temp_DF = pd.DataFrame()
        temp_DF['Date'] = temp

In [8]: dataframe = pd.merge(df_Predictions, temp_DF,
                           on='Date', how='right')

In [9]: dataframe[dataframe.isna().any(axis=1)]
        dates_empty = dataframe[dataframe.isna().any(axis=1)].reset_index() ['Date']

In [10]: pred_Temp =pd.DataFrame()

         for i in range (0, len(dates_empty)):
             row = df.loc[df['From'] == dates_empty[i]]
             pred_Temp = pred_Temp.append(row.tail(1))

         pred_Temp = pred_Temp.rename(index=str,
                                      columns = {'From':'Date'})

         pred_Temp = pred_Temp.drop(['Timestamp'],
                                    axis=1)

         dataframe = pred_Temp.set_index('Date').reindex_like(
             dataframe.set_index('Date')).combine_first(
             dataframe.set_index('Date')).reset_index()

         dataframe.shape
```

```
In [11]: dataframe = pd.merge(dataframe, df_Clearsky, on='Date', how='right')
         dataframe = dataframe.sort_values (by='Date')
         dataframe = dataframe.ffill()
         dataframe = dataframe.reset_index(drop=True)

         dataframe.shape

In [12]: normalSkyDatapoints = pd.read_table(
             'weather2Years.csv',
             sep=',' )

         normalSkyDatapoints = normalSkyDatapoints.drop(
             ['Record', 'Date',  'Time',
              'UVA_Avg', 'UVB_Avg', 'Tracker2WM_Avg',
              'WindDir_D1_WVT', 'WindDir_SD1_WVT',
              'ShadowbandWM_Avg', 'RH','AirTC_Avg',
              'WS_ms_S_WVT', 'BP_mB_Avg', 'DNICalc_Avg'],
             axis = 1)

         normalSkyDatapoints=normalSkyDatapoints.rename(
             index = str,
             columns = {'SunWM_Avg':'GHI',
                  'TrackerWM_Avg':'DNI',
                  'ShadowWM_Avg':'DHI',
                  'TimeStamp':'Date'})

         normalSkyDatapoints['Date'] = pd.to_datetime(
             normalSkyDatapoints['Date'])

         normalSkyDatapoints = normalSkyDatapoints.reset_index(
             drop=True)

In [13]: start = normalSkyDatapoints.iloc[0]['Date']
         end = normalSkyDatapoints.iloc[-1]['Date']

         normalSky = pd.DataFrame()
         normalSky['Date'] = pd.date_range(start, end, freq='min')
         normalSkyDF = pd.merge(normalSkyDatapoints, normalSky,
                                on = 'Date', how = 'outer')
         normalSkyDF = normalSkyDF.sort_values(by=['Date'])
         normalSkyDF = normalSkyDF.fillna(method='ffill')

In [14]: total = getTotalIrradiance(
             normalSkyDatapoints['GHI'],
             normalSkyDatapoints['DNI'],
             normalSkyDatapoints['DHI'],
             pd.DatetimeIndex(normalSkyDatapoints['Date']))

         normalSkyDF['Insolation_NS'] = total['poa_global']

         normalSkyDF=normalSkyDF.drop(
             ['DNI', 'DHI', 'GHI'],
```

```
        axis=1)

    dataframe=pd.merge(dataframe,
                normalSkyDF, on=['Date'])

    dataframe = dataframe.drop(['Degrees',
                                'Fog',
                                'Dew Temperature',
                                'Pressure',
                                'Windspeed'],
                               axis=1)

    loadprofileDF = pd.read_csv('2yearLoadProfile.csv', sep=';')
    loadprofileDF=loadprofileDF.rename(index=str,
                                       columns={'Time':'Date',
                                                'Sum [kWh]':'Load'
                                                })
    loadprofileDF['Date']=pd.to_datetime(loadprofileDF['Date'])
    loadprofileDF=loadprofileDF.drop(['Electricity.Timestep'],
                                     axis=1)

    dataframe=pd.merge(dataframe,
                loadprofileDF, on=['Date'])
```

In [15]:
```
    dataframe['PV Power'] = calcPVPower(
        dataframe['Insolation_NS'])

    dataframe = dataframe.set_index('Date')

    timeInterval = '{}T'.format(interval)

    dataframe = dataframe.resample(timeInterval).sum()

    dataframe = dataframe.reset_index()

    dataframe['Temperature'] = normaliseInterval(dataframe['Temperature'])
    dataframe['Humidity'] = normaliseInterval(dataframe['Humidity'])
    dataframe['Clouds'] = normaliseInterval(dataframe['Clouds'])
    dataframe['Low Clouds'] = normaliseInterval(dataframe['Low Clouds'])
    dataframe['Medium Clouds'] = normaliseInterval(dataframe['Medium Clouds'])
    dataframe['High Clouds'] = normaliseInterval(dataframe['High Clouds'])
    dataframe['Insolation_CS'] = normaliseInterval(dataframe['Insolation_CS'])
    dataframe['Insolation_NS'] = normaliseInterval(dataframe['Insolation_NS'])

    dataframe['Month'] = dataframe['Date'].dt.month
    dataframe['HourOfDay'] = dataframe['Date'].dt.hour

    dataframe['Battery']   = pd.Series(np.zeros(dataframe.shape[0]))
    dataframe['Generator'] = pd.Series(np.zeros(dataframe.shape[0]))

    dataframe['DayofWeek'] = dataframe['Date'].dt.weekday
```

```
        dataframe['Weekday'] =dataframe['DayofWeek'].apply(lambda x: 1 if x <= 4 else 0)

        dataframe['Month'] = pd.Categorical(dataframe['Month'])
        dataframe['Month'] = dataframe.Month.cat.codes

        dataframe['HourOfDay'] = pd.Categorical(dataframe['HourOfDay'])
        dataframe['HourOfDay'] = dataframe.HourOfDay.cat.codes

        dataframe['Weekday'] = pd.Categorical(dataframe['Weekday'])
        dataframe['Weekday'] = dataframe.Weekday.cat.codes

        dataframe = dataframe.drop(['DayofWeek'], axis=1)
```

```
In [16]: for i in range(1, len(dataframe)):
             dataframe.loc[i, 'Prev CS'] = dataframe.loc[i-1,
                                                         'Insolation_CS']

         for i in range(0, len(dataframe)-1):
             dataframe.loc[i, 'Next CS'] = dataframe.loc[i+1,
                                                         'Insolation_CS']
```

```
In [17]: dataframe = dataframe[['Load', 'PV Power', 'Month', 'HourOfDay',
                               'Battery', 'Generator', 'Weekday',
                               'Temperature', 'Humidity', 'Clouds',
                               'Low Clouds', 'Medium Clouds', 'High Clouds',
                               'Insolation_CS', 'Insolation_NS', 'Next CS',
                               'Prev CS', 'Date']]
```

```
In [18]: dataframe.to_csv('{}DatabaseIoT_Batch2.csv'.format(interval))
```

```
In [19]: # df = pd.read_csv('15DatabaseIoT_Batch2.csv')
         # df = df.drop(['Unnamed: 0'], axis=1)

         # df['Date'] = pd.to_datetime(df['Date'])

         # dataframe = df.copy()
         # dataframe.head()
```

```
In [20]: df = dataframe.copy()
```

```
In [21]: df.head()
```

C.5.4 Train, test and validate datasets

```
In [22]: for i in range(0, len(df)-1):
             df.loc[i, 'Next PV'] = df.loc[i+1, 'PV Power']
             df.loc[i, 'Next Load'] = df.loc[i+1, 'Load']
```

```
In [23]: df_train_set = pd.DataFrame()
         df_valid_set = pd.DataFrame()
```

```
df_test_set = pd.DataFrame()

for j in range (1,13):
    year = 2019
    month = j
    train_day_start = 1
    train_day_end = 22
    valid_day_start = 23
    valid_day_end = 26
    test_start = 27
    test_end = 0

    days_31 = [1, 3, 5, 7, 8, 10, 12]
    days_30 = [4, 6, 9, 11]

    # February
    if j == 2:
        test_end = 28
    # 31: January, March, May, July, Aug, Oct, Dec
    if j in days_31:
        test_end = 31
    if j in days_30:
        test_end = 30

     # TRAIN DATA SEGMENT
    start_date = pd.Timestamp(year=year, month=month,
                              day=train_day_start,
                              hour = 0, minute = 0,
                              second = 0)
    end_date = pd.Timestamp(year=year, month=month,
                            day=train_day_end, hour = 23,
                            minute = 59, second = 59)

    dti = pd.date_range(start = start_date,
                        end = end_date, freq='15T')
    df_train_set = df_train_set.append(
        df.loc[df.Date.isin(pd.DatetimeIndex(dti))])

    # VALID DATA SEGMENT
    start_date = pd.Timestamp(year=year, month=month,
                              day=valid_day_start,
                              hour = 0, minute = 0,
                              second = 0)
    end_date = pd.Timestamp(year=year, month=month,
                            day=valid_day_end, hour = 23,
                            minute = 59, second = 59)

    dti = pd.date_range(start = start_date,
                        end = end_date, freq='15T')
    df_valid_set = df_valid_set.append(
        df.loc[df.Date.isin(pd.DatetimeIndex(dti))])
```

```
            # TEST DATA SEGMENT
            start_date = pd.Timestamp(year=year, month=month,
                                      day=test_start, hour = 0,
                                      minute = 0, second = 0)
            end_date = pd.Timestamp(year=year, month=month,
                                    day=test_end, hour = 23,
                                    minute = 59, second = 59)

            dti = pd.date_range(start = start_date,
                                end = end_date, freq='15T')

            df_test_set = df_test_set.append(
                df.loc[df.Date.isin(pd.DatetimeIndex(dti))])
```

```
In [24]: # print('Train:\t\t{0}\t{1:.1f}%'.format(
         #     df_train_set.shape[0],
         #     100*df_train_set.shape[0]/df.shape[0]))
         # print('Validate:\t{0}\t{1:.1f}%'.format(
         #     df_valid_set.shape[0],
         #     100*df_valid_set.shape[0]/df.shape[0]))
         # print('Test:\t\t{0}\t{1:.1f}%'.format(
         #     df_test_set.shape[0],
         #     100*df_test_set.shape[0]/df.shape[0]))
```

```
In [25]: df_train_set = df_train_set.drop(['Date'], axis=1)
         df_valid_set = df_valid_set.drop(['Date'], axis=1)
         df_test_set = df_test_set.drop(['Date'], axis=1)
```

```
In [26]: # df_train_set.to_csv('RL IoT Train.csv')
         # df_valid_set.to_csv('RL IoT Valid.csv')
         # df_test_set.to_csv('RL IoT Test.csv')
```

```
In [27]: df_train_set = pd.read_csv('RL IoT Train.csv')
         df_train_set = df_train_set.drop(['Unnamed: 0'], axis=1)

         df_valid_set = pd.read_csv('RL IoT Valid.csv')
         df_valid_set = df_valid_set.drop(['Unnamed: 0'], axis=1)

         df_test_set = pd.read_csv('RL IoT Test.csv')
         df_test_set = df_test_set.drop(['Unnamed: 0'], axis=1)
```

```
In [28]: train_stats=df_train_set.describe()
         train_stats=train_stats.transpose()

         valid_stats=df_valid_set.describe()
         valid_stats=valid_stats.transpose()

         test_stats=df_test_set.describe()
         test_stats=test_stats.transpose()
```

```
In [29]: def norm(x):
             return (x - train_stats['mean'])/train_stats['std']

         normed_Train = norm(df_train_set)
         normed_Valid = norm(df_valid_set)
         normed_Test = norm(df_test_set)

In [30]: df_train_set['Humidity'] = normed_Train['Humidity'].copy()
         df_train_set['Clouds'] = normed_Train['Clouds'].copy()
         df_train_set['Temperature'] = normed_Train['Temperature'].copy()
         df_train_set['Low Clouds'] = normed_Train['Low Clouds'].copy()
         df_train_set['Medium Clouds'] = normed_Train['Medium Clouds'].copy()
         df_train_set['High Clouds'] = normed_Train['High Clouds'].copy()
         df_train_set['Insolation_CS'] = normed_Train['Insolation_CS'].copy()
         df_train_set['Insolation_NS'] = normed_Train['Insolation_NS'].copy()
         df_train_set['Next CS'] = normed_Train['Next CS'].copy()

         df_valid_set['Humidity'] = normed_Valid['Humidity'].copy()
         df_valid_set['Clouds'] = normed_Valid['Clouds'].copy()
         df_valid_set['Temperature'] = normed_Valid['Temperature'].copy()
         df_valid_set['Low Clouds'] = normed_Valid['Low Clouds'].copy()
         df_valid_set['Medium Clouds'] = normed_Valid['Medium Clouds'].copy()
         df_valid_set['High Clouds'] = normed_Valid['High Clouds'].copy()
         df_valid_set['Insolation_CS'] = normed_Valid['Insolation_CS'].copy()
         df_valid_set['Insolation_NS'] = normed_Valid['Insolation_NS'].copy()
         df_valid_set['Next CS'] = normed_Valid['Next CS'].copy()

         df_test_set['Humidity'] = normed_Test['Humidity'].copy()
         df_test_set['Clouds'] = normed_Test['Clouds'].copy()
         df_test_set['Temperature'] = normed_Test['Temperature'].copy()
         df_test_set['Low Clouds'] = normed_Test['Low Clouds'].copy()
         df_test_set['Medium Clouds'] = normed_Test['Medium Clouds'].copy()
         df_test_set['High Clouds'] = normed_Test['High Clouds'].copy()
         df_test_set['Insolation_CS'] = normed_Test['Insolation_CS'].copy()
         df_test_set['Insolation_NS'] = normed_Test['Insolation_NS'].copy()
         df_test_set['Next CS'] = normed_Test['Next CS'].copy()

In [31]: train_set = df_train_set.as_matrix()
         valid_set = df_valid_set.as_matrix()
         test_set = df_test_set.as_matrix()

In [32]: action_perm  = {'use_pv': {True, False},
                        'charge_bat': {True, False},
                        'discharge_bat': {True, False},
                        'use_grid': {True, False},
                        'switch_on_Gen': {True, False}}

         # remove actions charge + discharge == True
         import itertools
         l = ['T', 'F']
         actions_all = [list(i) for i in itertools.product(l, repeat=5)]
```

```
        remove_ = 8

        te = 0

        while True:
            for i in range (0, len(actions_all)):
                if actions_all[i][1] == 'T' and actions_all[i][2] == 'T':
                    actions_all.remove(actions_all[i])
                    te += 1
                    break
            if te == remove_:
                break

        actions = []
        for i in range (0, len(actions_all)):
            actions.append(actions_all[i][0] + actions_all[i][1] \
                           + actions_all[i][2]+ actions_all[i][3] \
                           + actions_all[i][4])
```

```
In [33]: num_actions=len(actions)
```

```
In [34]: num_states = 17
```

```
In [35]: def step(action, t, prev_state, data):

             next_state    = None

             action        = list(action)
             use_pv        = action[0]
             charge_bat    = action[1]
             discharge_bat = action[2]
             use_grid      = action[3]
             use_gen       = action[4]

             # Load and PV for the time step
             PV    = data[t][17]
             load  = data[t][18]

             # Where the battery and generator are starting
             bat   = prev_state[4]
             gen   = prev_state[5]

             bat_before = bat

             grid  = grid_interval # kWh
             LPS   = load #100%
             maxGen = gen_interval #kWh

             PV_Use  = 0
             Gen_Use = 0
             Grid_Use = 0
             Bat_Use = 0
```

```
availablePV = 0
availableGrid = 0
availableGen = 0

############################
# Use PV Action
if use_pv == 'T':
    availablePV = PV
    if availablePV > load:
        PV_Use += LPS
        availablePV -= LPS
        LPS = 0
    else:
        LPS -= availablePV
        PV_Use += availablePV
        availablePV = 0
############################
# Discharge Battery Action
if discharge_bat == 'T':
    if LPS < bat:
        bat -= LPS
        Bat_Use += LPS
        LPS = 0
    else:
        LPS -=bat
        Bat_Use += bat
        bat = 0
############################
# Use Grid Action
if use_grid == 'T':
    availableGrid = grid
    if LPS < availableGrid:
        availableGrid -= LPS
        Grid_Use += LPS
        LPS = 0
    else:
        LPS -= availableGrid
        Grid_Use += availableGrid
        availableGrid = 0
############################
# Use Generator Action
if use_gen == 'T':
    availableGen = maxGen
    if gen < (15/interval):
        availableGen = 0.2*availableGen
    if LPS < availableGen:
        availableGen -= LPS
        Gen_Use += LPS
        LPS = 0
    else:
        LPS -= availableGen
        Grid_Use += availableGen
```

```
        availableGen = 0
############################
# Charge batteries Action
#maxC = MaxCharge_kWh
if charge_bat == 'T':
    if use_pv == 'T':
        if bat+availablePV > fully_charged_kWh:
            PV_Use += fully_charged_kWh - bat
            bat = fully_charged_kWh
        else:
            bat += availablePV
            PV_Use += availablePV
    if use_grid == 'T':
        if bat+availableGrid > fully_charged_kWh:
            Grid_Use += fully_charged_kWh - bat
            bat = fully_charged_kWh
        else:
            bat += availableGrid
            Grid_Use += availableGrid
    if use_gen == 'T':
        if bat+availableGen > fully_charged_kWh:
            Gen_Use += fully_charged_kWh - bat
            bat = fully_charged_kWh
        else:
            bat += availableGen
            PV_Use += availableGen
############################

# Rewards
reward = 0
reward = reward + int(LPS <= 0)*20
reward = reward + int(PV_Use == PV and use_pv == 'T')*6
reward = reward + int(PV_Use > 0)*6
reward = reward + int(use_gen == 'T' and PV == 0)*3
reward = reward + int(discharge_bat == 'T' and PV == 0)*6

if use_gen == 'F':
    gen = 0 # switch off generator
else:
    gen += 1

battery_charged = bat - bat_before
battery_charged = max(0, battery_charged)

LPS = max(0, LPS)

# THE NEXT STATE
next_state    = [data[t][18],
                 data[t][17],
                 data[t][2],
                 data[t][3],
                 bat,
```

```
                                gen,
                                data[t][6],
                                data[t][7],
                                data[t][8],
                                data[t][9],
                                data[t][10],
                                data[t][11],
                                data[t][12],
                                data[t][13],
                                data[t][14],
                                data[t][15],
                                data[t][16]]

            if t+1 == len(data):
                next_state = None

            return next_state, reward, t+2 == len(
                data), LPS, PV_Use, Gen_Use, Grid_Use, Bat_Use, battery_charged, None
```

In [36]:
```
# number of episodes
num_episodes  = 50
```

In [37]:
```
memory = []

max_memory = int(2*len(train_set))

def sampleMemory(batch_size):
    if batch_size > len(memory):
        return random.sample(memory,
                             (len(memory)))
    else:
        return random.sample(memory,
                             batch_size)

def addSampleMemory(memorySample):
    memory.append(memorySample)
    if len(memory) > max_memory:
        memory.pop(0)
```

In [38]:
```
max_epsilon = 1
min_epsilon = 0.01

Lambda = 0.001
gamma  = 0.7

batch_hours = 3
batch_size  = int((60/interval)*batch_hours)
```

In [39]:
```
def build_model():
    model = keras.Sequential(
    [layers.Dense(32, activation=tf.nn.relu,
```

```
                    input_shape=(num_states,)),
             layers.Dense(32, activation=tf.nn.relu),
             layers.Dense(32, activation=tf.nn.relu),
             layers.Dense(num_actions)
            ])

            return model

In [40]: model = build_model()

         model.compile(loss='mse',
                       optimizer=tf.keras.optimizers.Adam(
                           lr=Lambda))

         print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                576
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 24)                792
=================================================================
Total params: 3,480
Trainable params: 3,480
Non-trainable params: 0
_________________________________________________________________
None
```

```
In [41]: total_LPS = np.zeros(num_episodes)
         total_rewards = np.zeros(num_episodes)
         total_PV = np.zeros(num_episodes)
         total_Gen = np.zeros(num_episodes)
         total_Grid = np.zeros(num_episodes)
         total_BatU = np.zeros(num_episodes)
         total_BatC = np.zeros(num_episodes)
```

C.5.5 Training

```
In [42]: def RLController_Train():
             epsilon  = max_epsilon
             cnt      = 0
             steps    = 0

             data = train_set
```

```
while cnt < num_episodes:
    print('Episode {}'.format(cnt))
    # reset state to start of dataset
    state = [data[0][0],
             data[0][1],
             data[0][2],
             data[0][3],
             data[0][4],
             data[0][5],
             data[0][6],
             data[0][7],
             data[0][8],
             data[0][9],
             data[0][10],
             data[0][11],
             data[0][12],
             data[0][13],
             data[0][14],
             data[0][15],
             data[0][16]]
    state[4] = initial_charge
    state[5] = 0 #off

    # save episode
    episode_rewards = 0
    episode_LPS = 0
    episode_PV = 0
    episode_Gen = 0
    episode_Grid = 0
    episode_BatU = 0
    episode_BatC = 0

    # run entire episode
    for t in range(len(data)):
        action=0

        if random.random() < epsilon:
            action = random.randint(0, num_actions - 1)
        else:
            options = model.predict(np.array([state]))
            action = np.argmax(options)

        # step through data
        next_state, reward, done, LPS, PV, Gen, Grid, BatU, BatC, _ = step(
            actions[action], t, state, data)

        addSampleMemory(
            [state, action, reward, next_state])

        random_batch = sampleMemory(batch_size)

        states_ = np.array(
```

```
            [val[0] for val in random_batch])

    next_states = np.array(
            [(np.zeros(num_states)
              if val[3] is None else val[3]) for val in random_batch])

    q_s_a = model.predict(states_)

    q_s_a_d = model.predict(next_states)

    x_batch = np.zeros((len(random_batch),
                            num_states))

    y_batch = np.zeros((len(random_batch),
                            num_actions))

    for i, b in enumerate(random_batch):
        state_B, action_B, reward_B, next_state_B = b[0], b[1], b[2], b[3]

        current_q_B = q_s_a[i]

        if next_state_B == None:
            current_q_B[action_B] = reward_B
        else:
            current_q_B[action_B]=reward_B + gamma*np.amax(
                q_s_a_d[i])

        x_batch[i] = state_B
        y_batch[i] = current_q_B

    model.fit(x=x_batch, y=y_batch, verbose=0)

    steps +=1

    state = next_state

    epsilon = min_epsilon + (max_epsilon-min_epsilon)*math.exp(
        -Lambda*steps)

    episode_rewards += reward
    episode_LPS += LPS
    episode_PV += PV
    episode_Gen += Gen
    episode_Grid += Grid
    episode_BatU += BatU
    episode_BatC += BatC

    if done:
        total_rewards[cnt]=episode_rewards
        total_LPS[cnt]=episode_LPS
        total_PV[cnt]=episode_PV
        total_Gen[cnt]=episode_Gen
```

```
                        total_Grid[cnt] = episode_Grid
                        total_BatU[cnt] = episode_BatU
                        total_BatC[cnt] = episode_BatC
                        break

                cnt += 1
```

```
In [43]: RLController_Train()
```

```
In [44]: model.save('model IoT.h5')
```

```
In [45]: plt.plot(total_rewards)
         plt.xlabel('Episodes')
         plt.ylabel('Total rewards obtained per episode during training')
         # plt.show()
         plt.savefig('Training_rewards_IoT.pdf',bbox_inches = 'tight')
```

```
In [46]: plt.plot(total_LPS)
         plt.xlabel('Episodes')
         plt.ylabel('Total LPS [kWh] per episode during training')
         #plt.show()
         plt.savefig('Training_LPS_IoT.pdf',bbox_inches = 'tight')
```

```
In [47]: plt.plot(total_PV)
         plt.plot(total_BatU)
         plt.plot(total_Gen)
         plt.plot(total_Grid)
         plt.xlabel('Episodes')
         plt.ylabel('Total Power [kWh] per episode during training')
         plt.legend(['PV', 'Battery', 'Generator', 'Grid'])
         #plt.show()
         plt.savefig('Training_HPS.pdf',bbox_inches = 'tight')
```

C.5.6 Validation

```
In [48]: validate_Baseline1  = []
         validate_Baseline2  = []
         validate_RL = []
```

```
In [49]: def baselineController1(data, Process):
             cnt = 0
             steps = 0

             # reset state to start of dataset
             state    = [data[0][0],
                         data[0][1],
                         data[0][2],
                         data[0][3],
                         data[0][4],
                         data[0][5],
```

```
                data[0][6],
                data[0][7],
                data[0][8],
                data[0][9],
                data[0][10],
                data[0][11],
                data[0][12],
                data[0][13],
                data[0][14],
                data[0][15],
                data[0][16]]

    state[4] = initial_charge
    state[5] = 0 # off

    episode_rewards = 0
    episode_LPS = 0
    episode_PV = 0
    episode_Gen = 0
    episode_Grid = 0
    episode_BatU = 0
    episode_BatC = 0

    # run entire episode
    for t in range(len(data)):
        # random action
        action = random.randint(0, num_actions - 1)

        # step through data
        next_state, reward, done, LPS, PV, Gen, Grid, BatU, BatC,_ = step(
            actions[action], t, state, data)

        steps +=1

        state = next_state

        episode_LPS += LPS
        episode_rewards += reward
        episode_PV += PV
        episode_Gen += Gen
        episode_Grid += Grid
        episode_BatU += BatU
        episode_BatC += BatC

        if done:
            if Process == 'Validate':
                global validate_Baseline1
                validate_Baseline1 = [episode_rewards,
                                      episode_LPS,
                                      episode_BatU,
                                      episode_PV,
                                      episode_Gen,
```

```
                                            episode_Grid,
                                            episode_BatC]

                    if Process == 'Test':
                        global test_Baseline1
                        test_Baseline1 = [episode_rewards,
                                          episode_LPS,
                                          episode_BatU,
                                          episode_PV,
                                          episode_Gen,
                                          episode_Grid,
                                          episode_BatC]
                    break
```

```
In [50]: def baselineController2(data, Process):
             cnt = 0
             steps = 0

             # reset state to start of dataset
             state    = [data[0][0],
                         data[0][1],
                         data[0][2],
                         data[0][3],
                         data[0][4],
                         data[0][5],
                         data[0][6],
                         data[0][7],
                         data[0][8],
                         data[0][9],
                         data[0][10],
                         data[0][11],
                         data[0][12],
                         data[0][13],
                         data[0][14],
                         data[0][15],
                         data[0][16]]

             state[4] = initial_charge
             state[5] = 0 #off

             episode_rewards = 0
             episode_LPS = 0
             episode_PV = 0
             episode_Gen = 0
             episode_Grid = 0
             episode_BatU = 0
             episode_BatC = 0

             for t in range(len(data)):

                 # rule-based action
                 use_pv        = None
```

```
charge_bat    = None
discharge_bat = None
use_grid      = None
use_gen       = None

# use PV
if state[1] > 0:
    use_pv = 'T'
else:
    use_pv = 'F'
# charge bat if PV > (load)
if state[1] > state[0]:
    charge_bat = 'T'
else:
    charge_bat = 'F'
# discharge bat if PV < load
if state[1] < state[0] and charge_bat != 'T':
    discharge_bat = 'T'
else:
    discharge_bat = 'F'
# use grid randomly
if bool(random.getrandbits(1)) == 1:
    use_grid = 'T'
else:
    use_grid = 'F'
# use generator randomy
if bool(random.getrandbits(1)) == 1:
    use_gen = 'T'
else:
    use_gen = 'F'

action = use_pv+charge_bat+discharge_bat+use_grid+use_gen

# step through data
next_state, reward, done, LPS, PV, Gen, Grid, BatU, BatC,_ = step(
    action, t, state, data)

steps += 1

state = next_state

episode_rewards += reward
episode_LPS += LPS
episode_PV += PV
episode_Gen += Gen
episode_Grid += Grid
episode_BatU += BatU
episode_BatC += BatC

if done:
    if Process == 'Validate':
        global validate_Baseline2
```

```
                    validate_Baseline2 = [episode_rewards,
                                          episode_LPS,
                                          episode_BatU,
                                          episode_PV,
                                          episode_Gen,
                                          episode_Grid,
                                          episode_BatC]

                if Process == 'Test':
                    global test_Baseline2
                    test_Baseline2 = [episode_rewards,
                                      episode_LPS,
                                      episode_BatU,
                                      episode_PV,
                                      episode_Gen,
                                      episode_Grid,
                                      episode_BatC]
                break
```

```
In [51]: def RLController(data, Process):
             epsilon  = max_epsilon
             cnt      = 0
             steps    = 0

             # reset state to start of dataset
             state    = [data[0][0],
                         data[0][1],
                         data[0][2],
                         data[0][3],
                         data[0][4],
                         data[0][5],
                         data[0][6],
                         data[0][7],
                         data[0][8],
                         data[0][9],
                         data[0][10],
                         data[0][11],
                         data[0][12],
                         data[0][13],
                         data[0][14],
                         data[0][15],
                         data[0][16]]
             state[4] = initial_charge
             state[5] = 0 #off

             # save episode
             episode_rewards = 0
             episode_LPS = 0
             episode_PV = 0
             episode_Gen = 0
             episode_Grid = 0
             episode_BatU = 0
```

```
episode_BatC = 0

if Process == 'Test':
    global model
    del model
    model = keras.models.load_model('model IoT.h5')

# run entire episode
for t in range(len(data)):
    action=0

    if random.random() < epsilon:
        action = random.randint(0, num_actions - 1)
    else:
        options = model.predict(np.array([state]))
        action = np.argmax(options)

    # step through data
    next_state, reward, done, LPS, PV, Gen, Grid, BatU, BatC, _ = step(
        actions[action], t, state, data)

    addSampleMemory(
        [state, action, reward, next_state])

    random_batch = sampleMemory(batch_size)

    states_ = np.array(
            [val[0] for val in random_batch])

    next_states = np.array(
            [(np.zeros(num_states)
              if val[3] is None else val[3]) for val in random_batch])

    q_s_a = model.predict(states_)

    q_s_a_d = model.predict(next_states)

    x_batch = np.zeros((len(random_batch),
                            num_states))

    y_batch = np.zeros((len(random_batch),
                            num_actions))

    for i, b in enumerate(random_batch):
        state_B, action_B, reward_B, next_state_B = b[0], b[1], b[2], b[3]

        current_q_B = q_s_a[i]

        if next_state == None:
            current_q_B[action_B] = reward_B
        else:
            current_q_B[action_B]=reward_B + gamma*np.amax(
```

```
                        q_s_a_d[i])

                    x_batch[i] = state_B
                    y_batch[i] = current_q_B

                model.fit(x=x_batch, y=y_batch, verbose=0)

                steps +=1

                state = next_state

                epsilon = min_epsilon + (max_epsilon-min_epsilon)*math.exp(
                    -Lambda*steps)

                episode_rewards += reward
                episode_LPS += LPS
                episode_PV += PV
                episode_Gen += Gen
                episode_Grid += Grid
                episode_BatU += BatU
                episode_BatC += BatC

                if done:
                    if Process == 'Validate':
                        global validate_RL
                        validate_RL = [episode_rewards,
                                       episode_LPS,
                                       episode_BatU,
                                       episode_PV,
                                       episode_Gen,
                                       episode_Grid,
                                       episode_BatC,]

                    if Process == 'Test':
                        global test_RL
                        test_RL = [episode_rewards,
                                   episode_LPS,
                                   episode_BatU,
                                   episode_PV,
                                   episode_Gen,
                                   episode_Grid,
                                   episode_BatC,]

                cnt += 1
```

```
In [52]: baselineController1(valid_set, 'Validate')
```

```
In [53]: baselineController2(valid_set, 'Validate')
```

```
In [54]: RLController(valid_set, 'Validate')
```

```
In [55]: validationResults = pd.DataFrame(columns=['Simulation',
                                                   'Rewards',
                                                   'LPS',
                                                   'BatU',
                                                   'PV',
                                                   'Gen',
                                                   'Grid',
                                                   'BatC'])

         totalLoad = df_valid_set['Load'].sum()

         B1_validationResults = {'Simulation': 'Baseline1',
                                 'Rewards': validate_Baseline1[0],
                                 'LPS': validate_Baseline1[1],
                                 'BatU': validate_Baseline1[2],
                                 'PV': validate_Baseline1[3],
                                 'Gen': validate_Baseline1[4],
                                 'Grid': validate_Baseline1[5],
                                 'BatC': validate_Baseline1[6]}

         B2_validationResults = {'Simulation': 'Baseline2',
                                 'Rewards': validate_Baseline2[0],
                                 'LPS': validate_Baseline2[1],
                                 'BatU': validate_Baseline2[2],
                                 'PV': validate_Baseline2[3],
                                 'Gen': validate_Baseline2[4],
                                 'Grid': validate_Baseline2[5],
                                 'BatC': validate_Baseline2[6]}

         RL_validationResults = {'Simulation': 'RL',
                                 'Rewards': validate_RL[0],
                                 'LPS': validate_RL[1],
                                 'BatU': validate_RL[2],
                                 'PV': validate_RL[3],
                                 'Gen': validate_RL[4],
                                 'Grid': validate_RL[5],
                                 'BatC': validate_RL[6]}

         validationResults = validationResults.append(
             B1_validationResults, ignore_index=True)
         validationResults = validationResults.append(
             B2_validationResults, ignore_index=True)
         validationResults = validationResults.append(
             RL_validationResults, ignore_index=True)

         print('Total Load {}'.format(df_valid_set['Load'].sum()))

         usedB1 = validate_Baseline1[2] + validate_Baseline1[3] \
             + validate_Baseline1[4] + validate_Baseline1[5]
         chargedB1 = validate_Baseline1[6]
         netB1 = totalLoad - usedB1 - validate_Baseline1[1] + chargedB1
         print('Net B1 {} '.format(netB1))
```

```
        usedB2 = validate_Baseline2[2] + validate_Baseline2[3] \
                + validate_Baseline2[4] + validate_Baseline2[5]
        chargedB2 = validate_Baseline2[6]
        netB2 = totalLoad - usedB2 - validate_Baseline2[1] + chargedB2
        print('Net B2 {} '.format(netB2))

        usedRL = validate_RL[2] + validate_RL[3] + validate_RL[4] + validate_RL[5]
        chargedRL = validate_RL[6]
        netRL = totalLoad - usedRL - validate_RL[1] + chargedRL
        print('Net RL {} '.format(netRL))
```

```
Total Load 641.366897313
```

```
In [56]: validationResults
```

```
Out[61]:   Simulation Rewards         LPS        BatU          PV        Gen  \
        0  Baseline1   56912  134.460557  152.335297  256.129713  80.437152
        1  Baseline2   74042    7.415123  294.376042  584.163509   7.134757
        2         RL   76326   40.571909  265.794858  478.085239  31.985183

                Grid        BatC
        0  170.350222  152.335297
        1   40.071571  291.783358
        2   87.921287  262.980832
```

C.5.7 Testing

```
In [57]: test_Baseline1 = []
         test_Baseline2 = []
         test_RL = []
```

```
In [58]: baselineController1(test_set, 'Test')
         baselineController2(test_set, 'Test')
```

```
In [59]: RLController(test_set, 'Test')
```

```
In [60]: testResults = pd.DataFrame(columns=['Simulation',
                                             'Rewards',
                                             'LPS',
                                             'BatU',
                                             'PV',
                                             'Gen',
                                             'Grid',
                                             'BatC'])

         totalLoad = df_test_set['Load'].sum()

         B1_testResults = {'Simulation': 'Baseline1',
```

```
                    'Rewards': test_Baseline1[0],
                    'LPS': test_Baseline1[1],
                    'BatU': test_Baseline1[2],
                    'PV': test_Baseline1[3],
                    'Gen': test_Baseline1[4],
                    'Grid': test_Baseline1[5],
                    'BatC': test_Baseline1[6]}

B2_testResults = {'Simulation': 'Baseline2',
                    'Rewards': test_Baseline2[0],
                    'LPS': test_Baseline2[1],
                    'BatU': test_Baseline2[2],
                    'PV': test_Baseline2[3],
                    'Gen': test_Baseline2[4],
                    'Grid': test_Baseline2[5],
                    'BatC': test_Baseline2[6]}

RL_testResults = {'Simulation': 'RL',
                    'Rewards': test_RL[0],
                    'LPS': test_RL[1],
                    'BatU': test_RL[2],
                    'PV': test_RL[3],
                    'Gen': test_RL[4],
                    'Grid': test_RL[5],
                    'BatC': test_RL[6]}

testResults = testResults.append(
    B1_testResults, ignore_index=True)
testResults = testResults.append(
    B2_testResults, ignore_index=True)
testResults = testResults.append(
    RL_testResults, ignore_index=True)

print('Total Load {}'.format(df_valid_set['Load'].sum()))

usedB1 = test_Baseline1[2] + test_Baseline1[3] \
        + test_Baseline1[4] + test_Baseline1[5]
chargedB1 = test_Baseline1[6]
netB1 = totalLoad - usedB1 - test_Baseline1[1] + chargedB1
print('Net B1 {} '.format(netB1))

usedB2 = test_Baseline2[2] + test_Baseline2[3] \
        + test_Baseline2[4] + test_Baseline2[5]
chargedB2 = test_Baseline2[6]
netB2 = totalLoad - usedB2 - test_Baseline2[1] + chargedB2
print('Net B2 {} '.format(netB2))

usedRL = test_RL[2] + test_RL[3] + test_RL[4] + test_RL[5]
chargedRL = test_RL[6]
netRL = totalLoad - usedRL - test_RL[1] + chargedRL
print('Net RL {} '.format(netRL))
```

```
Total Load 641.366897313


In [61]: testResults

Out[86]:   Simulation Rewards         LPS        BatU          PV         Gen  \
         0  Baseline1   66316  147.742191  203.234009  329.617139  110.835237
         1  Baseline2   84778   12.145119  363.993464  692.501727    9.113292
         2         RL   87845   58.602098  328.497322  558.333560   34.829455

                Grid        BatC
         0  181.499179  203.020624
         1   52.123188  359.969659
         2  114.155410  324.510713
```

Bibliography

[1] U. E. I. Administration, "International energy outlook 2016 with projections to 2040," Online, 2016. [Online]. Available: https://www.eia.gov/outlooks/ieo/

[2] G. Zabel. (2009, April) Peak people: The interrelationship between population growth and energy resources. Resilience. Originally posted in Energy Bulletin. [Online]. Available: https://www.resilience.org/stories/2009-04-20/peak-people-interrelationship-between-population-growth-and-energy-resources/

[3] K. V. Tangirale, "Solar wind, hybrid renewable energy systems," *IEEE India*, vol. 12, no. 2, pp. 44–53, 2017.

[4] M. Kapfudza, N. Moorosi, and M. Thinyane, "GA optimization of hybrid energy systems for telecommunications in marginalised rural areas," in *2014 IEEE 6th International Conference on Adaptive Science Technology (ICAST)*, Oct 2014, pp. 1–6.

[5] A. Gonzalez, J. Riba, and A. Rius, "Optimal sizing of a hybrid grid-connected photovoltaic-wind-biomass power system," *Sustainability*, vol. 7, no. 9, pp. 1–20, 2015.

[6] F. Daniel and A. Rix, "Optimising the design of a hybrid power supply using a genetic algorithm," 01 2019, pp. 269–274.

[7] A. H. Shahirinia, S. M. M. Tafreshi, A. H. Gastaj, and A. R. Moghaddomjoo, "Optimal sizing of hybrid power system using genetic algorithm," in *2005 International Conference on Future Power Systems*, Nov 2005, pp. 6 pp.–6.

[8] N. Newman. (2019) Hybrid power plants. [Online]. Available: https://www.eniday.com/en/technology_en/hybrid-power-plants/

[9] South African Government. (2019) Energy. [Online]. Available: https://www.gov.za/about-sa/energy

[10] Department of Energy. (2019) The renewable energy data and information service. [Online]. Available: http://redis.energy.gov.za/

[11] Southern Africa Universities Radiometric Network. (2019) Solar radiometric data for the public. [Online]. Available: http://www.sauran.net/Data

[12] Solargis. (2019) Solar resource maps of South Africa. [Online]. Available: https://solargis.com/maps-and-gis-data/download/south-africa

[13] GeoSUN Africa. (2019) South African GIS layers. [Online]. Available: http://geosun.co.za/downloads/south-african-gis-layers/

[14] R. Luna, M. Perea, D. Vargas-Vázquez, and G. Moreno, "Optimal sizing of renewable hybrids energy systems: A review of methodologies," *Solar Energy*, vol. 86, pp. 1077–1088, 04 2012.

[15] M. Han, X. Zhang, L. Xu, R. May, S. Pan, and J. Wu, "A review of reinforcement learning methodologies on control systems for building energy," Dalarna University, Microdata Analysis, Tech. Rep. 2018:02, 2018.

[16] M. S. Ramli, S. S. A. Wahid, and K. K. Hassan, "A comparison of renewable energy technologies using two simulation softwares: HOMER and RETScreen," in *International Conference on Applied Physics and Engineering*, 2017.

[17] R. Dufo-López and J. L. Bernal-Agustín, "Design and control strategies of PV-diesel systems using genetic algorithms," *Solar Energy*, vol. 79, no. 1, pp. 33 – 46, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0038092X04003020

[18] HOMER. (2019) HOMER. [Online]. Available: https://www.homerenergy.com/

[19] M. M. Z. Tanim, N. Chowdhury, M. Rahman, and J. Ferdous, "Design of a photovoltaic-biogas hybrid power generation system for Bangladeshi remote area using HOMER software," in *2014 3rd International Conference on the Developments in Renewable Energy Technology (ICDRET)*, 05 2014, pp. 1–5.

[20] K. Sopian, A. Zaharim, Y. Ali, Z. M. Nopiah, J. A. Razak, and N. S. Muhammad, "Optimal operational strategy for hybrid renewable energy system using genetic algorithms," *WSEAS Transaction on Mathematics*, vol. 7, no. 4, pp. 130–140, 2008.

[21] A. Kumar, M. Zaman, N. Goel, N. Goel, and R. Church, "In search of an optimization tool for renewable energy resources: HOMER vs. in-house model," in *2013 IEEE Electrical Power Energy Conference*, 2013, pp. 1–7.

[22] Y. A. Katsigiannis, P. Georgilakis, and E. S. Karapidakis, "Genetic algorithm solution to optimal sizing problem of small autonomous hybrid power systems," in *Artificial Intelligence: Theories, Models and Applications: 6th Hellenic Conference on AI*, May 2010, pp. 327–332.

[23] R. Atia and N. Yamada, "Optimization of a pv-wind-diesel system using a hybrid genetic algorithm," in *Electrical Power and Energy Conference (EPEC), 2012 IEEE*. IEEE, October 2012, pp. 80–85.

[24] M. Ko, Y. S. Kim, M. H. Chung, and H. C. Jeon, "Multi-objective optimization design for a hybrid energy system using the genetic algorithm," *Energies*, vol. 8, no. 4, pp. 2924–2949, 2015.

[25] D. Xu, L. Kang, L. Chang, and B. Cao, "Optimal sizing of standalone hybrid wind/PV power systems using genetic algorithms," in *Canadian Conference on Electrical and Computer Engineering, 2005.*, May 2005, pp. 1722–1725.

[26] Y. S. Zhao, J. Zhan, Y. Zhang, D. P. Wang, and B. G. Zou, "The optimal capacity configuration of an independent wind/PV hybrid power supply system based on improved PSO algorithm," in *8th International Conference on Advances in Power System Control, Operation and Management (APSCOM 2009)*, Nov 2009, pp. 1–7.

[27] L. Wang and C. Singh, "Multicriteria design of hybrid power generation systems based on a modified particle swarm optimization algorithm," *IEEE Transactions on Energy Conversion*, vol. 24, no. 1, pp. 163–172, March 2009.

[28] N. Mohd Razali and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving TSP," in *International Conference of Computational Intelligence and Intelligent Systems*, vol. 2, January 2011.

[29] T. Ma, C. R. Lashway, Y. Song, and O. Mohammed, "Optimal renewable energy farm and energy storate sizing methodologies for future hybrid power system," in *17th International Conference of Electrical Machines and Systems*. IEEE, October 2014, pp. 2827–2832.

[30] V. K. Soni and R. Khare, "Optimal sizing of HRES for small sized institute using HOMER," in *2014 IEEE 2nd International Conference on Electrical Energy Systems (ICEES)*, Jan 2014, pp. 77–81.

[31] A. Ali, "Model-free reinforcement learning for minimizing grid interaction of solar energy generation and heat pump loads in net zero-energy buildings," Master's thesis, Delft University of Technology, 2017.

[32] J. Cao and R. Xiong, "Reinforcement learning-based real-time energy management for plug-in hybrid electric vehicle with hybrid energy storage system," *Energy Procedia*, vol. 142, pp. 1876–6102, December 2017.

[33] B. Mbuwir, F. Ruelens, F. Spiessens, and G. Deconinck, "Battery energy management in a microgrid using batch reinforcement learning," *Energies*, vol. 10, no. 11, p. 1846, 2017.

[34] R. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2017.

[35] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Int. Res.*, vol. 4, no. 1, pp. 237–285, May 1996. [Online]. Available: http://dl.acm.org/citation.cfm?id=1622737.1622748

[36] R. Leo, R. S. Milton, and S. Sibi, "Reinforcement learning for optimal energy management of a solar microgrid," in *2014 IEEE Global Humanitarian Technology Conference - South Asia Satellite (GHTC-SAS)*, Sep 2014, pp. 183–188.

[37] Adventures in Machine Learning. (2019) Reinforcement learning tutorial with TensorFlow. [Online]. Available: https://adventuresinmachinelearning.com/reinforcement-learning-tensorflow/

[38] X. Qi, G. Wu, K. Boriboonsomsin, M. J. Barth, and J. Gonder, "Data-driven reinforcement learning–based real-time energy management system for plug-in hybrid electric vehicles," *Transportation Research Record*, vol. 2572, no. 1, pp. 1–8, 2016.

[39] R. Xiong, J. Cao, and Q. Yu, "Reinforcement learning-based real-time power management for hybrid energy storage system in the plug-in hybrid electric vehicle," *Applied energy*, vol. 211, pp. 538–548, 2018.

[40] L. A. Bollinger and R. Evins, "Multi-agent reinforcement learning for optimizing technology deployment in distributed multi-energy systems," in *23rd International Workshop of the European Group for Intelligent Computing in Engineering. Krakow, Poland*, 2016.

[41] S. Yue, D. Zhu, Y. Wang, and M. Pedram, "Reinforcement learning based dynamic power management with a hybrid power supply," in *2012 IEEE 30th International Conference on Computer Design (ICCD)*, Sept 2012, pp. 81–86.

[42] V. François-Lavet, D. Taralla, D. Ernst, and R. Fonteneau, "Deep reinforcement learning solutions for energy microgrids management," in *European Workshop on Reinforcement Learning (EWRL 2016)*, 2016.

[43] A. Rencher and G. Schaalje, *Linear Models in Statistics*. John Wiley and Sons, Inc., 2008.

[44] S. Dedgaonkar, V. Patil, N. Rathod, G. Hakare, and J. Bhosale, "Solar energy prediction using least square linear regression method," *International Journal of Current Engineering and Technology*, vol. 6, no. 5, pp. 1549–1552, September 2016.

[45] G. Masters, *Renewable and Efficient Electric Power Systems*, 2nd ed. Hoboken, New Jersey: John Wiley and Sons, Inc., 2013.

[46] Noah Pflugradt, "Load profile generator 8.6." [Online]. Available: www.loadprofilegenerator.de

[47] G. Rossum, "Python reference manual," 1995.

[48] F. Chollet, "Keras," 2015. [Online]. Available: https://keras.io

[49] J. Brownlee. (2019) A gentle introduction to the rectified linear unit (relu). [Online]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[50] ——. (2019) Gentle introduction to the adam optimization algorithm for deep learning. [Online]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[51] (2017) Return on investment. [Online]. Available: https://www.homerenergy.com/products/pro/docs/latest/return_on_investment.html

[52] W. F. Holmgren, C. W. Hansen, and M. A. Mikofski, "pvlib python: a python package for modeling solar energy systems," *Journal of Open Source Software*, vol. 3, no. 29, p. 884, 9 2018. [Online]. Available: http://dx.doi.org/10.21105/joss.00884

[53] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445. Austin, TX, 2010, pp. 51–56.

[54] Sustainable.co.za. (2019) SMA Sunny Tripower 15000TL inverter. [Online]. Available: https://www.sustainable.co.za/sma-sunny-tripower-15000tl-inverter.html

[55] ——. (2019) SMA Sunny Tripower 20000TL inverter. [Online]. Available: https://www.sustainable.co.za/sma-sunny-tripower-20000tl-inverter.html

[56] ——. (2019) SMA Sunny Tripower 25000TL inverter. [Online]. Available: https://www.sustainable.co.za/sma-sunny-tripower-25000tl-inverter.html

[57] The Power Store. (2019) OmniPower 120Ah 12V sealed battery. [Online]. Available: https://thepowerstore.co.za/products/omnipower-120ah-12v-sealed-battery

[58] ——. (2019) OmniPower 180Ah 12V sealed battery. [Online]. Available: https://thepowerstore.co.za/products/omnipower-180ah-12v-sealed-battery

[59] We Able. (2019) OmniPower 240Ah 12V sealed battery. [Online]. Available: https://www.weable.co.za/backup-power-solutions/omnipower-240ah-12v-sealed-battery

[60] Adendorft Machinery Mart. (2019) Mac-Afric 50 kVA (40 kW) standby silent diesel generator with ATS (380 V). [Online]. Available: https://www.adendorff.co.za/product/mac-afric-50-kva-40-kw-standby-silent-diesel-generator-ats-380v/

[61] ——. (2019) Mac-Afric 34 kVA (27.5 kW) prime power 380V silent type stand by generator. [Online]. Available: https://www.adendorff.co.za/product/mac-afric-42-kva-34-kw-prime-power-380v-silent-type-stand-by-generator/

[62] ——. (2019) MAC-AFRIC 37.5 kVA (30 KW) standby silent diesel generator with ATS (380V). [Online]. Available: https://www.adendorff.co.za/product/mac-afric-37-5-kva-30-kw-standby-silent-diesel-generator-ats-380v/

[63] Sustainable.co.za. (2019) Renewsys Galactic 380W solar panel. [Online]. Available: https://www.sustainable.co.za/renewsys-galactic-380w-solar-panel.html

[64] ——. (2019) Renewsys Galactic 365W solar panel. [Online]. Available: https://www.sustainable.co.za/renewsys-galactic-365w-solar-panel.html

[65] ——. (2019) JA Solar 330W Poly 5BB solar panel. [Online]. Available: https://www.sustainable.co.za/ja-solar-330w-poly-5bb-solar-panel.html

[66] The Power Store. (2019) Canadian Solar 330W super high power poly PERC HiKU. [Online]. Available: https://thepowerstore.co.za/products/canadian-solar-330w-super-high-power-poly-perc-hiku

[67] ——. (2019) Canadian Solar 405W super high power poly PERC HiKU with MC4. [Online]. Available: https://thepowerstore.co.za/collections/solar-panels/products/canadian-solar-405w-super-high-power-poly-perc-hiku-with-mc4

[68] ——. (2019) Canadian Solar 400W super high power poly PERC HiKU. [Online]. Available: https://thepowerstore.co.za/collections/solar-panels/products/canadian-solar-400w-super-high-power-poly-perc-hiku

[69] ——. (2019) Canadian Solar 410W super high power poly PERC HiKU with MC4. [Online]. Available: https://thepowerstore.co.za/collections/solar-panels/products/canadian-solar-410w-super-high-power-poly-perc-hiku-with-mc4

[70] ——. (2019) Canadian Solar 360W poly KuMax half-cell 35mm frame with MC4. [Online]. Available: thepowerstore.co.za/collections/solar-panels/products/canadian-solar-360w-poly-kumax-half-cell-35mm-frame

[71] ——. (2019) Canadian Solar 395W super high power poly PERC HiKU. [Online]. Available: https://thepowerstore.co.za/products/canadian-solar-395w-super-high-power-poly-perc-hiku

[72] University of Cape Town and individual contributors. (2017) Object-oriented programming in python documentation. [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/python-textbok/1.0/python-textbok.pdf

[73] The TensorFlow Authors and South Methodist University Center for Scientific Computation. (2018) Basic regression with keras via tensorflow. [Online]. Available: http://faculty.smu.edu/csc/workshops/2019/summer/hpc/tfk_basic_regression.html